7-23-1993

Automatic Tuning of Integrated Filters Using Neural Networks

Lutz Henning Lenz
Portland State University

AN ABSTRACT OF THE THESIS OF Lutz Henning Lenz for the Master of Science in Electrical Engineering presented July 23, 1993.

Title: Automatic tuning of integrated filters using Neural Networks

APPROVED BY THE MEMBERS OF THE THESIS COMMITTEE:

[redacted]

George G. Lendaris, Chair

[redacted]

Rolf Schaumann

[redacted]

Hormóz Zareh

Component values of integrated filters vary considerably due to manufacturing tolerances and environmental changes. Thus it is of major importance that the components of an integrated filter be electronically tunable. The method explored in this thesis is the transconductance-C-method.

A method of realizing higher-order filters is to use a cascade structure of second-order filters. In this context, a method of tuning second-order filters becomes important.

The research objective of this thesis is to determine if the Neural Network methodology can be used to facilitate the filter tuning process for a second-order filter (realized via the transconductance-C-method). Since this thesis is, at least to the knowledge of the author, the first effort in this direction, basic principles of filters and of Neural Networks [1-22] are presented.

A control structure is proposed which comprises three parts: the filter, the Neural Network, and a digital spectrum analyzer. The digital spectrum analyzer sends a test signal to the filter and measures the magnitude of the output at 49 frequency samples. The Neural Network part includes a memory that stores the 49 sampled values of the nominal spectrum. A comparator subtracts the latter values from the measured (actual) values, and feeds them as input to the Neural Network. The outputs of the Neural Network are the values of the percentage tuning amount. The adjusting device, which is envisioned as a component of the filter itself, translates the output of the Neural Network to adjustments in the value of the filter's transconductances.

Experimental results provide a demonstration that the Neural Network methodology can be usefully applied to the above problem context. A feedforward, single-hidden-layer Backpropagation Network reduces the manufacturing errors of up to 85% for the pole frequency and of up to 41% for the quality factor down to less than approximately 5% each. It is demonstrated that the method can be iterated to further reduce the error.

AUTOMATIC TUNING OF INTEGRATED FILTERS USING NEURAL NETWORKS

by

LUTZ HENNING LENZ

A thesis submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE
IN
ELECTRICAL ENGINEERING

Portland State University
1993

TO THE OFFICE OF GRADUATE STUDIES:

The members of the committee approve the thesis of Lutz Henning Lenz presented July 23, 1993.

George G. Lendaris, Chair

Rolf Schaumann

Hormoz Zarrh

APPROVED:

Rolf Schaumann, Chairman, Department of Electrical Engineering

Roy W. Koch, Vice Provost for Graduate Studies and Research

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER I

INTRODUCTION

The motivation for my thesis was the result of different, yet interrelating aspects. Schaumann [1] says : "All modern communication systems and most measuring equipment contain various types of electrical filters that the designer has to realize in an appropriate technology." Decisions about how a filter should be realized bring up questions such as which type of filter is to be used, or which order of filter is required, and questions about the implementation method. In general, the choice of network implementation is based on economic and technological considerations. As far as this thesis is concerned, technological considerations are emphasized. As is well known, without tuning, the transfer function of an integrated, continuous-time filter will vary considerably due to fabrication tolerances, environmental changes (temperature, humidity, aging) and parasitic effects. Thus it is of major importance that the components of an integrated filter be electronically tunable, and this implies an implementation approach based, for example, on the transconductance-C-method. This method is the main focus of the present thesis, but in addition passive, discrete RLC-networks are described and investigated, since they provide additional insight into the problem solving process.

Current manufacturing practice typically employs a "master and slave" tuning process [2,3,4]. This process involves manufacturing two copies of the same filter on one chip. An assumption is made that the two copies of the filter are identical -- i.e., the same manufacturing errors will have been made on both of them. The one called 'slave' is used to process the information-carrying signal, the one called 'master' is presented with a reference signal and its output is used to derive information for tuning the 'slave' filter

-- and tunes itself at the same time. In this way, tuning can be accomplished on-line; the cost associated with this is duplicate circuitry plus tuning circuitry on the (same) chip [2,3]. As with any tuning method, an accurate reference frequency is required. The accuracy of the "master and slave" tuning technique is limited by the accuracy of the actual matching of the characteristics of the two filters. Generally speaking, one can say that the existing (published) tuning methods do work, yet not as satisfactory as desired [1-7]. There is need for alternative methods to improve the tuning process.

The research objective of this thesis is to determine if the Neural Network (NN) methodology can be used to facilitate the filter tuning process and if the tuning circuitry required (the NN) can be implemented on a separate chip. Happily, the answer (to each question) turns out being yes.

CHAPTER II

FILTERS

II.1 INTRODUCTION

A filter may best be described as a two-port Circuit, which processes the magnitude and/or phase of an input signal in some prescribed way in order to generate a desired output signal.

In general, certain frequency components are transmitted (passed) by the filter with little or even no change, whereas other frequency components are rejected or stopped. Accordingly passbands (PB) and stopbands (SB) have been defined, as shown in Figure 1.

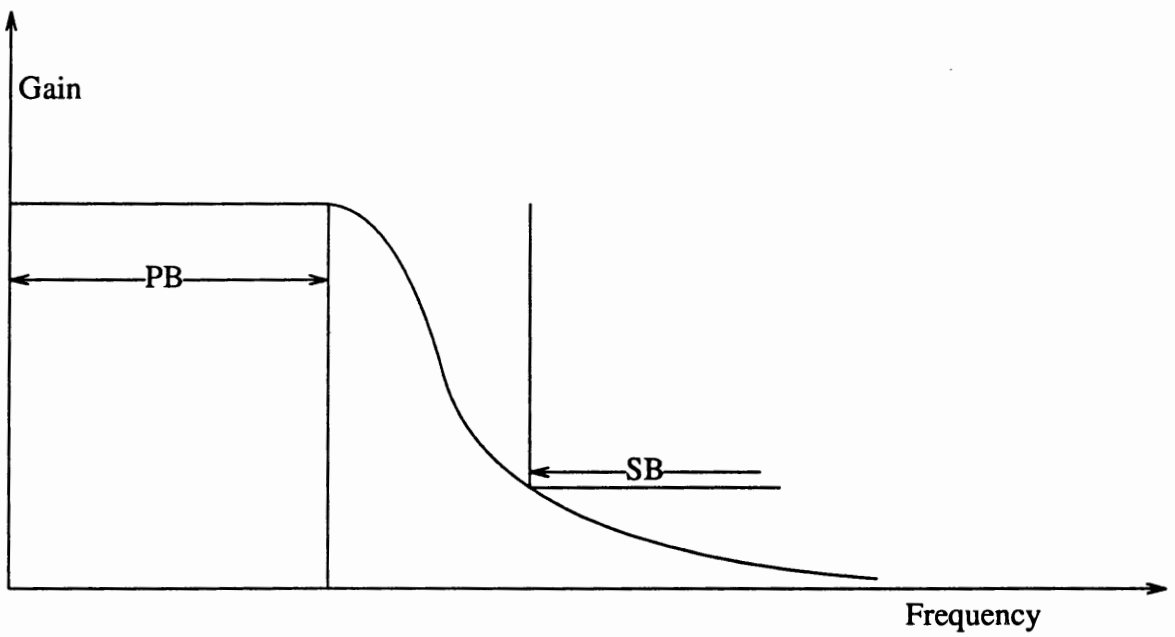


Figure 1. Passbands and stopbands.

This thesis does not consider the problem of designing a filter to realize a particular transfer function for some given problem. The interested reader can find many design techniques in the literature, e.g. [1] and [8]. The problem addressed is the following: measure the actual transfer function implemented by the manufactured filter, determine differences from the desired (nominal) transfer function, and calculate parameter values to perform a tuning of the manufactured filter.

II.2 PRELIMINARY CHARACTERIZATION OF FILTER TYPES

A common way to describe the general behavior of a filter is to characterize it by the frequency components it lets pass. Accordingly, a low-pass filter transmits low-frequency components and stops high-frequency components. Similarly, a high-pass filter transmits high-frequency components, and stops low-frequency components. Band-pass and band-stop filters may be described analogously. These filters, in contrast to low-pass and high-pass filters, stop or pass frequency components which are in an interval between high and low frequencies. As the title of this section suggests, this is only a general description of filter types, since the meaning of high-frequency and low-frequency is fuzzy. Their exact definition stays open, so far. As a consequence of the characterization given, we have four filter types: low-pass, high-pass, band-pass, and band-stop.

II.3 TRANSFER FUNCTIONS OF FILTERS

A given filter may be viewed as a black box (see Figure 2), whose input-output mapping is defined by the transfer function given in equation 2.1.

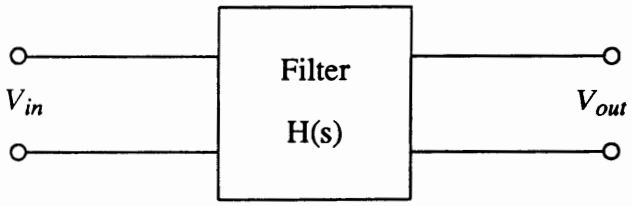


Figure 2. Input-output mapping of a filter.

$$H(s) = \frac{V_{out}}{V_{in}} = \frac{N(s)}{D(s)} = \frac{a_m s^m + \cdots + a_1 s + a_0}{s^n + b_{n-1} s^{n-1} + \cdots + b_1 s + b_0} \tag{2.1}$$

The order of the filter is given by the order of the denominator n, where $n \geq m$ has to be satisfied. The transfer function H(s) may be written in factored form [1] as:

$$H(s) = \frac{N(s)}{D(s)} = \frac{\prod_{k=1}^{m/2} (\alpha_{2k} s^2 + \alpha_{1k} s + \alpha_{0k})}{\prod_{i=1}^{n/2} (s^2 + s\,\omega_p / Q_p + \omega_p^{2})} \tag{2.2},$$

where the order of the filter n is assumed to be even. Equation (2.2) makes evident that a higher-order filter may be realized through a cascade of second-order filters.

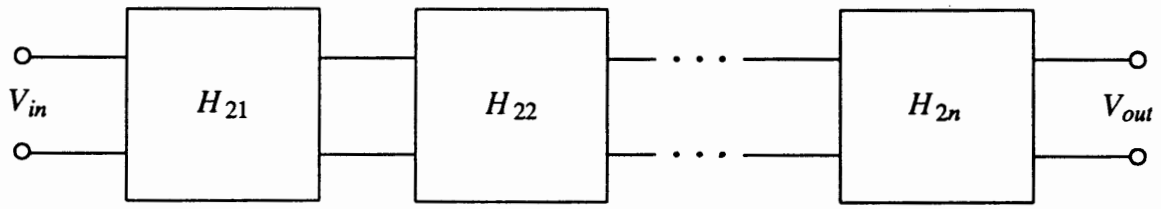


Figure 3. Higher-order filter realized through a cascade of second-order filters.($H_{2j}(s) = \frac{\alpha_{2j} s^2 + \alpha_{1j} s + \alpha_{0j}}{s^2 + s\,\omega_{pj} / Q_{pj} + \omega_{pj}^2}$).

If the order of the filter n is odd, either one first-order filter would have to be added or one third-order filter would have to replace one of the second-order filters in Figure 3.

Other realizations of higher-order filters, for example the ladder structure, are explained in [1]. However, as will be described in Section II.7, the approach taken in this

thesis is based on the cascade method of Figure 3.

II.4 CHARACTERIZATION OF SECOND-ORDER FILTER TYPES VIA TRANSFER FUNCTIONS

In general, the transfer function of a second-order filter may be written as

$$H_2(s) = \frac{a_2 s^2 + a_1 s + a_0}{s^2 + s\,\omega_p/Q_p + \omega_p^2} \qquad (2.3),$$

where ω_p represents the pole frequency and Q_p stands for the pole quality factor.

As mentioned in Section II.2, the behavior of a filter may be described as low-pass, high-pass, band-pass, or band-stop. Varying coefficients in equation (2.3) leads to each of the various types of filter. This is demonstarted in Table I. Simultaneously the transfer function given in this Table may be seen as a definition of the different filter types.

II.5 PASSIVE AND ACTIVE SECOND ORDER FILTER REALIZATIONS

Historically, passive, discrete RLC-networks have been used to implement filters. More recently, to reduce the size and cost of these networks, the large and costly inductors have been replaced by active networks. The result is active RC-networks composed of resistors, capacitors, and transistors, later augmented with operational amplifiers. Simultaneously, this enabled engineers to utilize advances in integrated Circuit technology to implement low-cost filters with small size. Nowdays, operational transconductance amplifiers (OTAs) are available that do not suffer from the restricted bandwidths that the initial active components such as operational amplifiers had. OTAs have significantly higher bandwidth than operational amplifiers [1]. A main disadvantage of such integrated filters is that they are more affected by variability of component values realized during manufacture, or due to environmental changes. Since active filters are

TABLE I

DEFINITION OF FILTER TYPES

Type of filter	Transfer function $H_2(s)$	Remarks
Low-Pass	$\frac{a_0}{s^2 + s\,\omega_p/Q_p + \omega_p^2}$	$a_2 = 0\,,\ a_1 = 0$ in some realizations: $a_0 = \omega_p^2$
Band-Pass	$\frac{a_1 s}{s^2 + s\,\omega_p/Q_p + \omega_p^2}$	$a_2 = 0\,,\ a_0 = 0$ in some realizations: $a_1 = \omega_p/Q_p$
High-Pass	$\frac{a_2 s^2}{s^2 + s\,\omega_p/Q_p + \omega_p^2}$	$a_1 = 0\,,\ a_0 = 0$ in some realizations: $a_2 = 1$
Band-Stop	$\frac{a_2 s^2 + a_0}{s^2 + s\,\omega_p/Q_p + \omega_p^2}$	$a_1 = 0$
All-Pass	$\frac{a_2 s^2 + a_1 s + a_0}{s^2 + s\,\omega_p/Q_p + \omega_p^2}$	$a_2 = 1$ $a_1 = -\omega_p/Q_p$ $a_0 = \omega_p^2$ *Magnitude* $= \lvert H_2(s)\rvert = 1$

electronically tunable, this turns out to be a problem that can be solved. Filters implemented using the transconductance-C-method can easily be tuned by bias currents or bias voltages. The ideal OTA (Figure 4.) is a voltage-controlled current source described by

$$I_0 = g_m (V^+ - V^-)$$

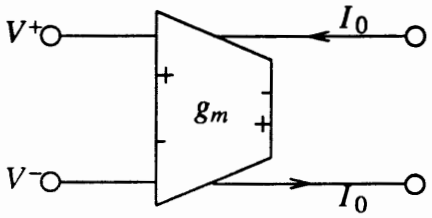


Figure 4. Transconductor circuit symbol.

In many designs, the value of the transconductance g_m is proportional to the value of a control bias current, and this bias current may be independently set.

As mentioned earlier, this thesis is concerned mainly with active filter realizations, but passive realizations are also discussed for development of ideas. An example for each is given in the following section. Both second-order filters realize the biquadratic function given in equation (2.3).

II.5.1 Example of a passive filter realization

Figure 5 shows a possible realization of a second-order low-pass RCL-filter.

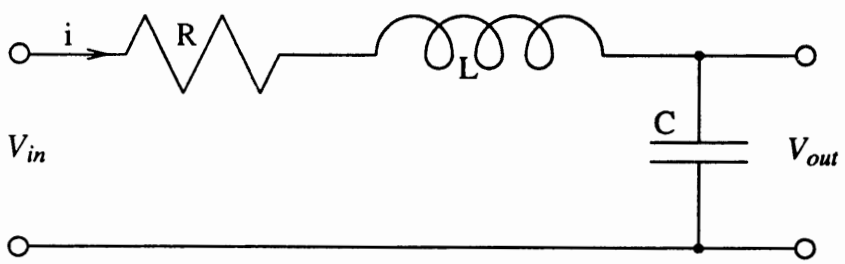


Figure 5. Second order low-pass RCL-Filter realization.

Input voltage: $V_{in} = V_R + V_L + V_C$

Output voltage: $V_{out} = V_c$

where: $V_R = IR$, $V_L = ILs$, $V_C = \frac{I}{Cs}$.

Using the above equations, the transfer function of the RCL-filter shown in Figure 5 is easily derived as

$$H_2(s) = \frac{V_{out}}{V_{in}} = \frac{1/LC}{s^2 + sR/L + 1/LC} \tag{2.4}$$

Comparing this with Table I, we see immediately that the condition $a_2 = a_1 = 0$ is satisfied, and thus, this filter realizes a second-order low-pass filter. We also see that $a_0 = \frac{1}{LC} = \omega_p^2$ as well as $Q_p = \omega_p \frac{L}{R} = \frac{1}{R}\sqrt{\frac{L}{C}}$.

II.5.2 Example of an active filter realization

Figure 6 shows the realization of a second-order transconductance-C-filter given in [2].

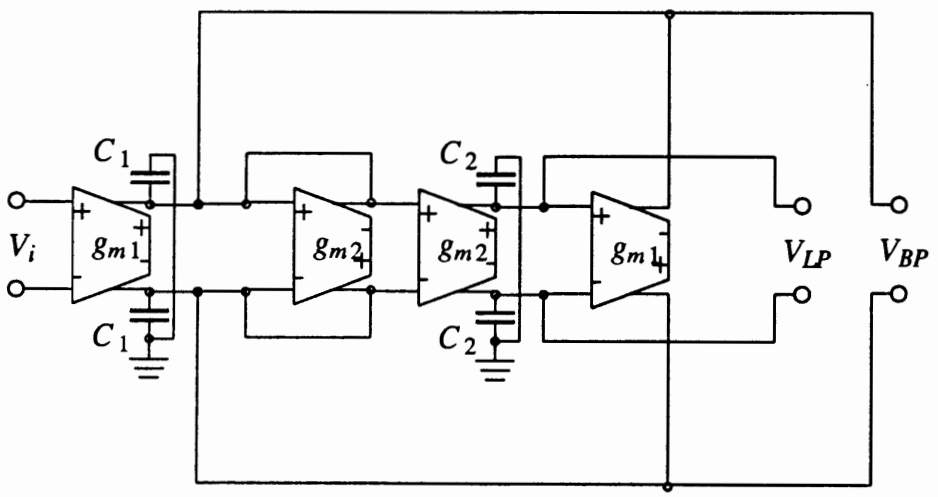


Figure 6. Second order g_m-C filter with low-pass and band-pass outputs.

The reader can verify by simple analysis that the circuit realizes the second-order low-pass transfer function

$$H_{LP} = \frac{V_{LP}}{V_i} = \frac{g_{m1}g_{m2}}{C_1C_2s^2 + sC_2g_{m2} + g_{m1}g_{m2}} = \frac{\omega_0^2}{s^2 + s\,\omega_0/Q_0 + \omega_0^2} \qquad (2.5).$$

Comparing this with Table I, we see immediately that the condition $a_2 = a_1 = 0$ is satisfied, and thus, this filter also realizes a second-order low-pass filter. In this case we see that $a_0 = \frac{g_{m1}g_{m2}}{C_1C_2} = \omega_p^2$ and $Q_p = \sqrt{\frac{g_{m1}C_1}{g_{m2}C_2}}$.

II.6 THE TUNING PROBLEM FOR CONTINUOUS-TIME INTEGRATED FILTERS

Without tuning, the transfer function of a continuous-time filter could vary considerably owing to fabrication tolerances, environmental changes (temperature, humidity, aging), and parasitic effects [1] (see Figure 7).

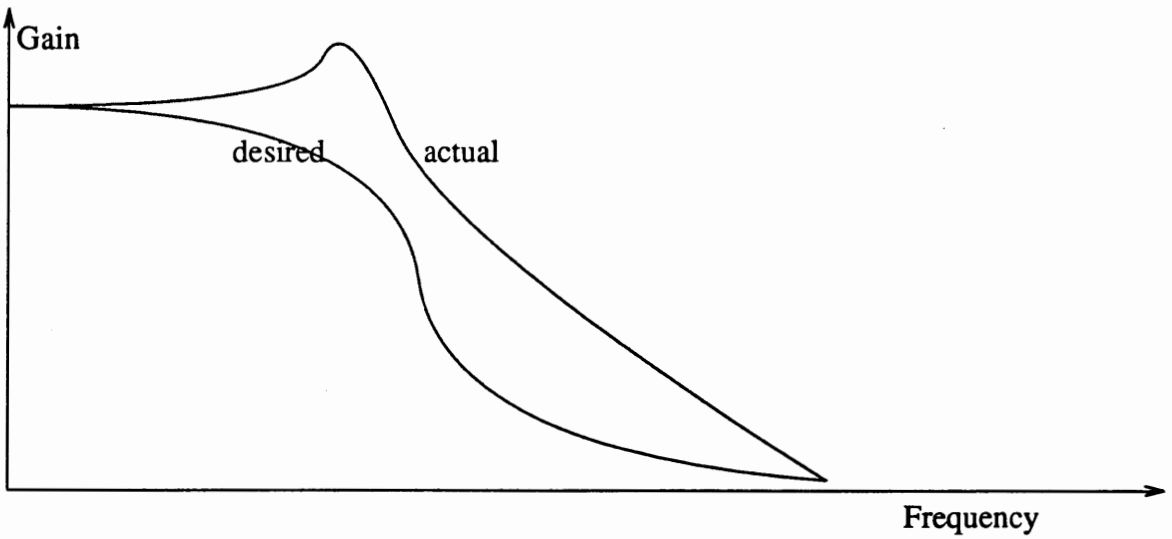


Figure 7. Nominal and actual transfer function.

To obtain accurate filter performance, the component values also have to be accurate. IC processing is only reliable in realizing accurate ratios of like components. For example, it is an empirical fact that the ratio of two capacitors C_1/C_2 stays within an

interval of as low as 0.1% of the nominal value. Yet, the tolerances of absolute values of Cs and g_ms may approach 30%. Therefore, frequency parameters, which are determined by absolute component values, may be off, such that the filter does not perform within specifications. Typically, statistical methods are used to describe the deviations of the component values [2]. A normal distribution centered around the nominal value and with a variance of 15% of the nominal value may be assumed to be reasonable (i.e., the expected value equals the nominal value). Figure 8 shows the probability density function (pdf) of a standard normal distribution. The term standard normal distribution implies that the expectation value equals zero. The common symbol σ for the variance is used.

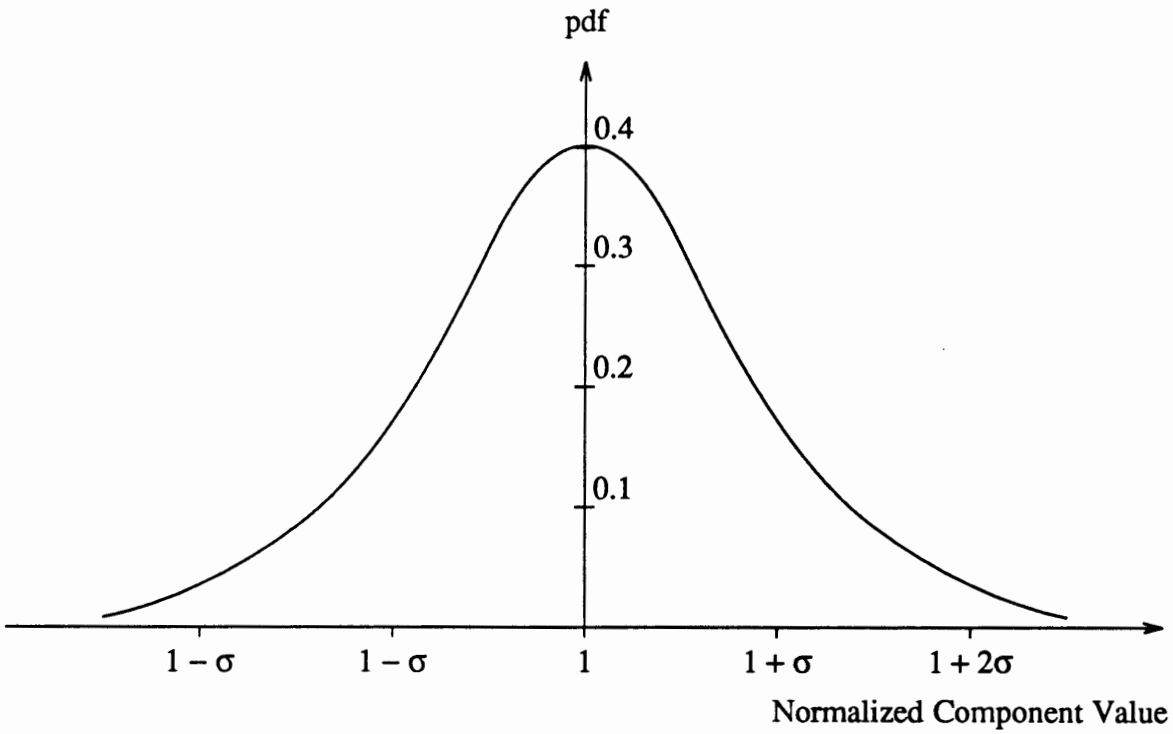


Figure 8. Normal distribution of component values.

As described in [1] and [2], tuning implies measuring filter performance, comparing it with a standard, calculating the error, and applying a correction to the filter to reduce the error by making use of a suitable control circuit. The problem definition,

given in the next section will reveal that the controller used in this approach will be realized as a NN, which provides the tuning parameter values off-line.

II.7 PROBLEM DEFINITION

Previous explanations discussed characteristics of filters, how they might be implemented and at which point difficulties may arise. Now we are in a position to define the problem to be solved in this thesis. The task of this thesis is to investigate if a NN is able to calculate the parameters for tuning a filter. Since this is, at least to the knowledge of the author, the first effort in this direction, there are several restrictions imposed. This thesis deals with second-order low-pass filters, in particular the filters shown in Figure 5 and in Figure 6, referred to as Circuit A and Circuit B, respectively, in the remainder of this thesis. This is not as severe a constraint as it might seem at first. In Section 2.3, it was explained that the building blocks for a higher-order filter are second-order filters (biquads) if the higher-order filter is realized in cascade structure. Table I illuminates that there is not a difference in principle between a low-pass, high-pass, or band-pass. Each numerator consists of one coefficient. The denominators even equal each other. Indeed a method, used in practice, is to design a low-pass filter and then, using an appropriate frequency transformation, to derive either a high-pass, band-pass, or band-stop filter. This is accomplished through the following scheme [2]: replace each independent frequency parameter s in the transfer function of a low-pass filter by $\frac{1}{s}$ ($\frac{1}{B}\frac{s^2+1}{s}$, $B\frac{s}{s^2+1}$), in order to obtain the transfer function of a high-pass (band-pass, band-stop) filter.

As far as the tuning problem is concerned, this thesis takes only the tuning of the magnitude of the transfer function into consideration. Thus the problem of tuning the filter to obtain the correct phase remains unsolved. This is the most severe restriction for applying any solution found in the forthcoming chapters. The task of simultaneously

tuning the phase and the magnitude has to be the subject of future research.

Furthermore for the purposes of the present research, it is assumed that the filter has already been designed. Through this design, we know both the nominal transfer function of the filter and the intended method of implementation. Circuit A and Circuit B show the two possibilities for implementation this thesis deals with - passive *RLC*-filters and active g_m–C-filters. As mentioned in the general introduction and in the discussion of the tuning problem, filter components vary. This is especially the case for integrated circuits, and hence, for filters implemented using the transconductance-*C* method. Yet, the main advantage is that these filters can be tuned electronically, by bias currents or bias voltages.

We have the following transfer function:

$$H_2(s) = \frac{\omega_p^2}{s^2 + s\,\omega_p/Q_p + \omega_p^2} \tag{2.6},$$

where, for Circuit A:

$$\omega_p = \sqrt{\frac{1}{LC}} \tag{2.7}$$

$$Q_p = \frac{1}{R}\sqrt{\frac{L}{C}} \tag{2.8}$$

and for Circuit B

$$\omega_p = \sqrt{\frac{g_{m1}g_{m2}}{C_1C_2}} \tag{2.9}$$

$$Q_p = \sqrt{\frac{g_{m1}C_1}{g_{m2}C_2}} \tag{2.10}.$$

The magnitude |H(jω)| derives to

$$|H(j\omega)| = \frac{1}{\sqrt{1 + (\frac{1}{Q_p^2} - 2)(\frac{\omega}{\omega_p})^2 + (\frac{\omega}{\omega_p})^4}} \tag{2.11}$$

Equations (2.7) through (2.11) show that with varying components - either g_{m1}, g_{m2}, C_1, C_2 or R, L, C - the magnitude as a function of the pole frequency and the pole

quality factor will alter. Different approaches for Circuit A and Circuit B are suggested in this thesis to tune the components of the manufactured filter to arrive at an implementation whose transfer function is close to the nominal values.

In Circuit A, all components have to be tuned, such that in the end, when the actual magnitude equals the nominal magnitude, each component (R,L,C) also takes on its nominal value (R_0,L_0,C_0). Since the magnitude (equation 2.11) is defined by only two parameters - Q_p and ω_p - tuning R, L, and C in Circuit A results in a non-unique inverse mapping. Different changes in R, L, and C cause the same error in the magnitude of the transfer function. Therefore additional information about the actual (manufactured) filter is required. For the case of Circuit A, this information is obtained through a dc measurement as described in Chapter IV.1, Data Generation. Simultaneously it will be sketched that the NN is able to determine the error terms ΔC and ΔL under the assumption that the resistor R takes on its nominal value, with no additional measurement needed.

In Circuit B, for considerations to be described, only the transconductances are allowed to be tuned to achieve a matching of the actual magnitude to the nominal magnitude. The logical reasoning behind this adopted constraint derives from the fact that the values of the two g_ms may be tuned by bias currents or bias voltages, in a continuous manner. On the other hand, the capacitors can only be changed in discrete steps and this process would require switches. Such switches would result in parasitic resistance and parasitic capacitance. These side-effects are, of course, not desired, as they would cause other difficulties. The empirical fact that the ratio of the capacitors C_1/C_2 stays constant within 0.1% during manufacturing allows us to do the tuning without modifying capacitor values.

It is important to realize that this choice - tuning only the g_ms, where the Cs stay untuned - is not really a constraint. The magnitude of the second-order low-pass filter (equation 2.11) depends only on two parameters. Therefore it ought to be enough to tune only two parameters to correct the magnitude, even if all of the four parameters are off.

Mathematically, this means that the g_ms are tuned such that they cancel out the errors caused by the capacitors. Also a dc measurement, as suggested for Circuit A, to make an unique determination, would not be practicable for Circuit B. In order to measure the value of a single component, the connections to this component would have to be cut. This is not a viable proposition.

For the present research, it is assumed that the (customer provided) nominal values are a pole frequency equal to 10 MHz and capacitors are available in the range between 0.1 pico-Farad and 20 pico-Farad. As a consequence, three different transfer functions for Circuit A and nine different transfer functions for Circuit B have been calculated. These transfer functions differ in the value of the quality factor Q_p. The values of the different nominal quality factors together with the component values are given in Tables II and III. Figure 9 shows how the magnitude changes for different quality factors.

As a consequence of the difficulties of current methods for tuning, the use of a Neural Network is suggested. This suggestion is based on the more general assumption that NNs are said to be 'smart' enough to learn non-linear relations. Additionally, several other requirements have to be fulfilled to make use of a NN in tuning the magnitude of a second-order low-pass filter. This will be described via Figure 10, the proposed control structure.

The filter output is sampled at 49 frequency components, centered around the nominal pole frequency ω_{p0}. The nominal magnitude is subtracted from the measured 'actual' magnitude. In practice, the measurement will be accomplished through a digital spectrum analyzer. The error-vector comprising the differences in the magnitude of the 49 frequency samples is the input for the NN. The error has been caused by the environment, where the expression environment stands as abbreviation for the physical environment during and after manufacturing.

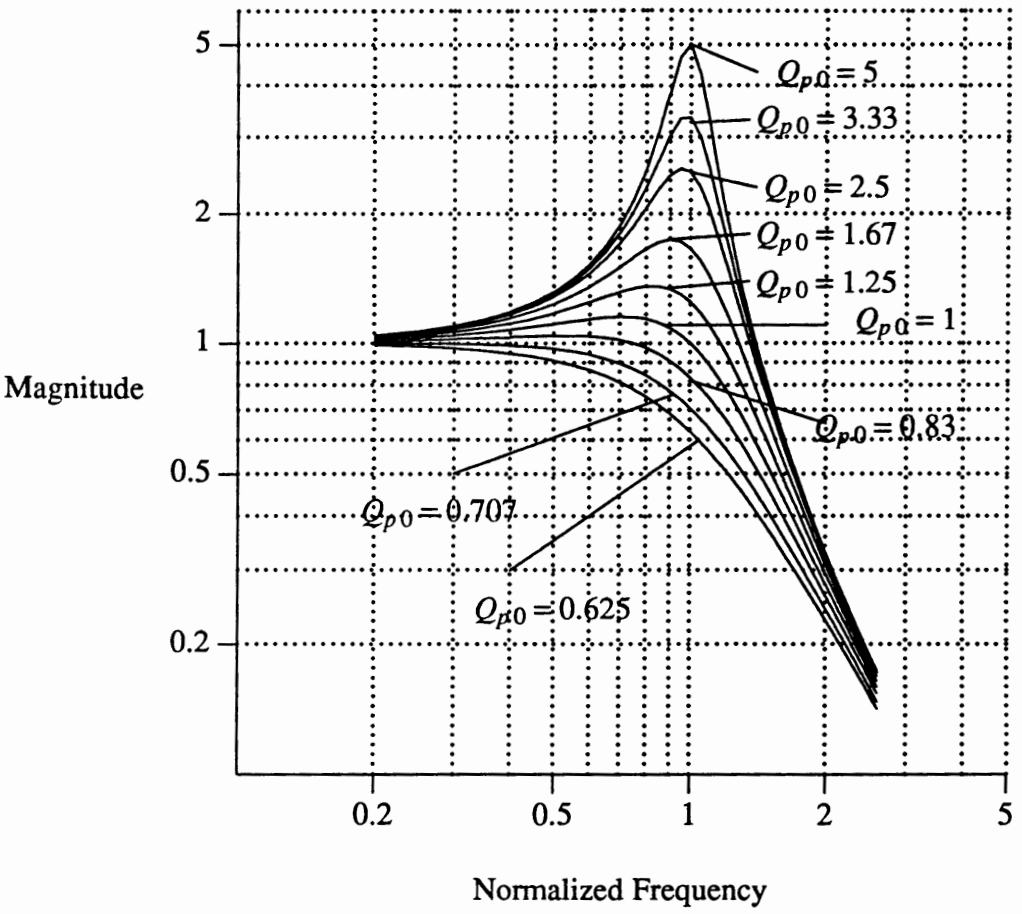


Figure 9. Magnitude of the transfer function for different quality factors Q_p.

TABLE II

NOMINAL QUALITY FACTORS AND COMPONENT VALUES OF CIRCUIT A

Q_{p0}	R in Ω	C in pF	L in mH
5	200	100	0.1
1	100	1000	0.01
0.625	160	1000	0.01

TABLE III

NOMINAL QUALITY FACTORS AND COMPONENT VALUES OF CIRCUIT B

Q_{p0}	g_{m10} in 10^{-5} S	g_{m20} in 10^{-5} S	C_{10} in pF	C_{20} in pF
5.0	2.5	2.5	12.5	0.5
3.33	1.6667	1.6667	5.5556	0.5
2.5	1.25	1.25	3.125	0.5
1.67	0.8333	0.8333	1.3889	0.5
1.25	0.625	0.625	0.7813	0.5
1.0	0.5	0.5	0.5	0.5
0.83	0.4167	0.4167	0.3472	0.5
0.707	0.3536	0.3536	0.25	0.5
0.625	0.3125	0.3125	0.1953	0.5

The number 49 has been a heuristic choice, yet based on practical considerations from a computational point of view. The latter comment refers to the fact that 49 data fill a 7x7 matrix. A matrix consisting of no more than seven columns fits in a Matlab diary file used to store the data for both training and testing. The following reasoning lead to the various choices made. The analytic expression of the transfer function is complete. The sampled representation becomes more and more sparse if fewer samples are taken. Based on human judgement, the decision was made that 49 frequency samples will represent the magnitude of the transfer function as a whole pattern. In fact, a question

that future research should answer has to be how many frequency samples are necessary and sufficient to represent the characteristics of the transfer function being considered. However, the results demonstrate that 49 frequency samples do a good job.

The NN is designed such that its outputs are the percentage of change the respective component of the filter has to be tuned to reduce the error towards zero. Therefore an adjusting device, which keeps track of the nominal value, is required. The task of this device is to multiply each of the outputs of the NN by the nominal value of the respective component of the filter and make the appropriate changes in the bias currents and/or voltages. Finally a memory to store the ideal or nominal spectrum sampled values of the magnitude of the transfer function is required.

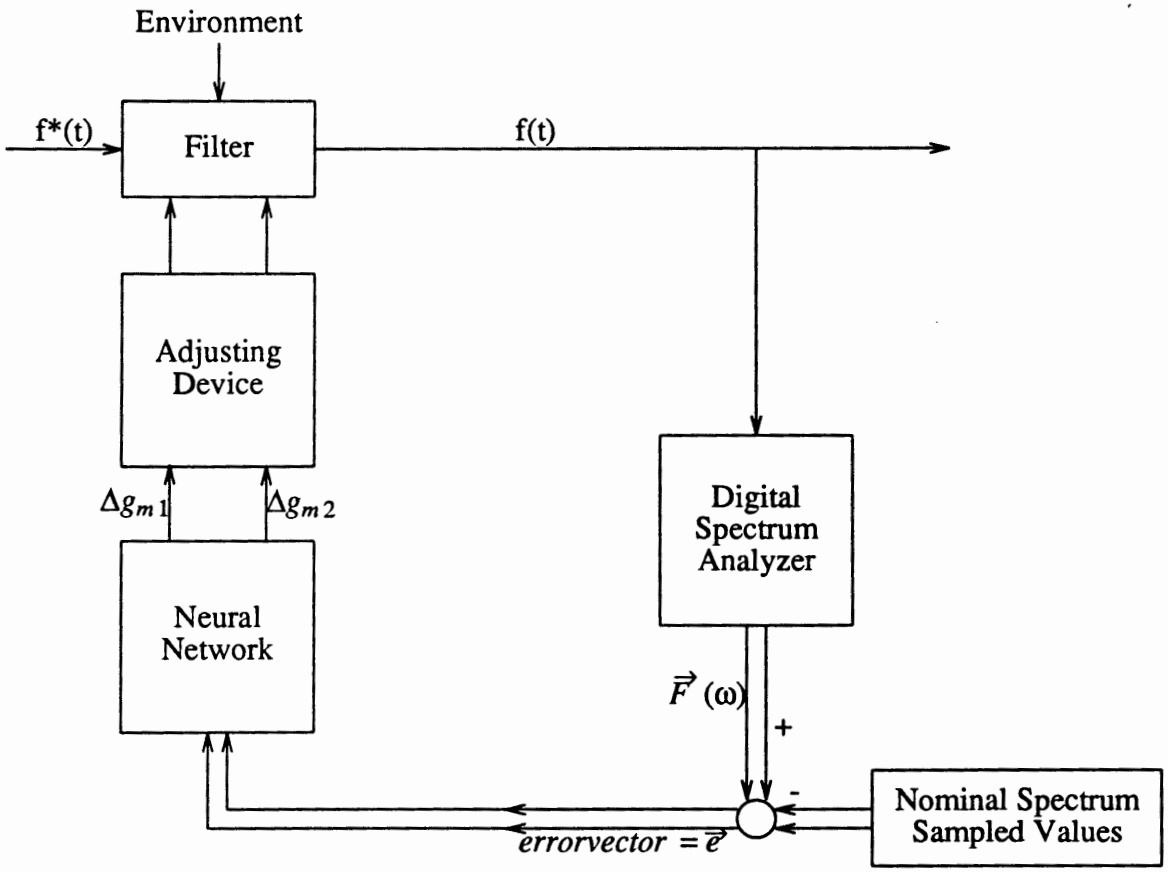


Figure 10. Control structure.

CHAPTER III

NEURAL NETWORKS

III.1 MOTIVATION FOR THE USE OF NEURAL NETWORKS

Feedforward NNs have been shown to learn unknown relationships even though only a limited number of example data pairs (input-output patterns) are used. Hence the NN may be described as robust. This means that the NN does not just memorize the training data, rather the trained NN has the ability to interpolate its output for input patterns not explicitly shown during training. The common expression for the latter behaviour is generalization. It is a question of major importance whether the NN being considered does a good job on the generalization or whether it does not.

In contrast to feedforward NNs, recurrent NNs include loops [9], which cause dynamics such as, for example, delay. For the given problem this is not required. The outputs of the NN, at a time, depend only on the current errorvector. In production mode, an input to the NN can be directly fed through the NN to the output. Feedback is required only during training, where the error of the output will be backpropagated to make adjustments within the NN. This step will be explained in Section III.3.

In general, the power of a NN-based approach does not necessarily lie in the elegance of a particular solution, but rather in the generality of the NN to find its own solution [9]. After a feasible approach to the problem being considered is found (this refers to the actual realization of the NN, e.g. components of the NN and their interrelations) the NN is simply presented with examples of the desired behavior or example data pairs. The significant advantage of a NN-based approach to problem solving is that we do not need to have a mathematical algorithm for mapping an input into an output. The

process of training is simply a matter of altering the connection weights systematically to encode the desired input-output relationship, which might be unknown. In our case, we construct the data sets by making a sequence of known changes in the circuit parameters, calculate the magnitude of the 49 sample points of the frequency response for the resulting transfer function, calculate the differences between these values and the corresponding nominal values, and use these differences as a 49-component error vector input to the NN. The NN will have as many outputs as we have parameters (e.g., Circuit A: ΔR, ΔL, ΔC; Circuit B: Δg_{m1}, Δg_{m2}). We know what Δ-values we used to generate the data, so these are used during the training process. The task of the NN is to learn to map a 49-component error vector to the correct Δ-values. (The author knows of no straightforward, analytical process to make such computations). A problem such as this is well suited to the Back-propagation-of-error method in the Neural Network technology [10].

III.2 BASIC BUILDING BLOCKS OF NEURAL NETWORKS

As described in [9], a NN structure might be described as a collection of parallel processors, in the following discussion referred to as processing elements (PEs). These PEs are connected together in the form of a directed digraph, and are organized such that the network structure lends itself to the problem being considered. Figure 11 schematically represents the PEs as nodes and the connection between the nodes as arcs. A graphical representation of one PE, is shown in Figure 12 and is described in Section III.2.1.

The specifics of an application determine the number of PEs for the input layer and for the output layer. In the context of filters, which need to be tuned, the number of PEs in the output layer (M) equals the number of components of the filter which are to be tuned. The number of input PEs (N) is determined by the number of frequency samples taken. The latter statement will be further illuminated in Section IV.1, data generation for a NN. The design of the (one or more) hidden layers is more complicated. In fact, this is a major part in the discussion of the NN experiments carried out.

After the NN has been designed, and after all NN parameters take on their initial values, the NN is prepared for the first phase of its life cycle, the training phase. During this phase, the NN is presented with example data pairs in order to adjust the weights of the connections. The testing phase follows. A criterion for evaluation, suitable for the problem context under consideration, has to be developed. As soon as the accuracy of the NN is sufficient, the last phase, which is the reason for doing all this, starts. The weights do not change, the NN is in production mode and fulfills its task.

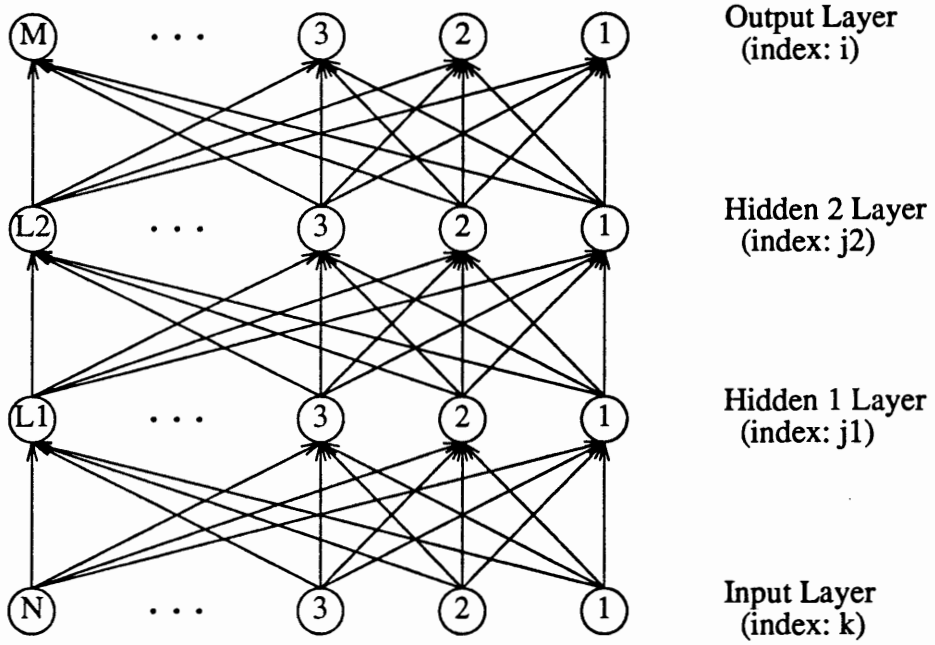


Figure 11. Layered feedforward Neural Network.

Realizing that NN structures are highly parallel simultaneously reveals that they ought to be and in fact are faster than commonly used sequential computers. What seems more important to the author, is the promising statement in [9], that NN architectures might be developed to solve problems belonging to the class of organized complexity. The latter expression is a term used by Hall and defined in [11, Chapter I]. Current research even led to the creation of new sciences of complexity [12].

III.2.1 Processing element

Inspired by knowledge from neuroscience, artificial neurons (alias processing elements) have been developed as the counterpart to biological neurons. As is well known, biological neurons comprise dentrites, synapses, an axon, the cell body (soma), and the nucleus. In analogy, the PE consists of several components, which are described below (with respect to the components of the biological neuron).

The model of the PE (Figure 12) replaces the tree-like network of the dentrites, which are connected to the cell body, by a so-called instar. Current theories assume that changes due to learning take place at the junction between two biological neurons, the synapse. A change in the neurotransmitter released by the presynaptic cell is asserted to be responsible [9]. The process of learning represented in nature may be best described as: the more often a process takes place, the stronger it gets. Since the basic theory comes from a book by Hebb [13], the literature [9,12,14] refers to this as Hebbian learning. The artificial neuron (PE) replaces these synapses by weights ($w_{i1} \ldots w_{iL1}$). During training, the weights of each PE are variable, such that the collection of PEs (NN) is in general capable of learning. How the process of learning takes place is described in the following section about the learning algorithm called Backpropagation. After training, the weights are fixed, with the exception that some learning paradigms exist, which allow the NN to learn on-line. In such cases, the weights adapt to the respective situation. Yet, the changes should be made only slowly to allow the NN to perform within specifications. The output of the PE symbolizes the axon. The cell-body, including the nucleus, may be seen represented by a threshold term and the non-linear function f. It is important to realize that the described relationship is only a principal characterization of similarities. One could argue in a different way by saying that in a biological neuron electrochemical processes take place, whereas in the PE all processes are of electrical nature. What seems most important to me, is to see that artificial neurons are not a new creation, rather they are based on biological neurons, which have been copied, in order to

create artificial learning processes.

The question which immediately arises is which task the PE has to perform. Each PE forms a weighted sum (equation 3.1) of the inputs (instar) from previous layers to which it is connected, adds a threshold value (equation 3.2) and produces a non-linear function f of this sum as its output value (equation 3.3). Depending on the problem context, the threshold value may be positive, zero or even negative. Since the non-linear function f (equation 3.3) converts the net input (equation 3.1) to an activation value for the respective PE, this function is called activation function. Another expression found in the literature is transfer function [15,16].

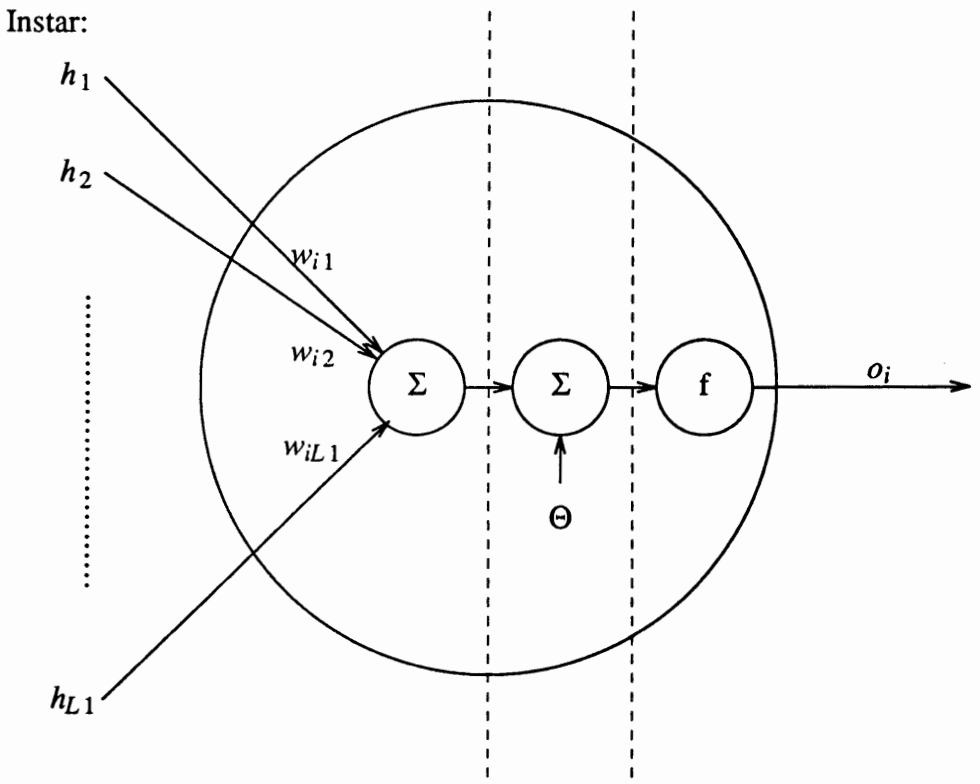


Figure 12. Processing element (PE).

$$net_i^{(p)} = \sum_{j=1}^{L1} w_{ij} h_j^{(p)} \tag{3.1}$$

$$net_i^{(p)*} = net_i^{(p)} + \theta_i = \sum_{j=1}^{L1} w_{ij} h_j^{(p)} + \theta_i \tag{3.2}$$

$$O_i^{(p)} = f\,(net_i^{(p)*}) = f\,(\sum_{j=1}^{L1} w_{ij} h_j^{(p)} + \theta_i) \tag{3.3}$$

The activation function has to be non-linear, to take advantage of the hidden layers. A linear activation function simply means that the output is proportional to the net input. Yet, this task is already accomplished through the weights. The most common transfer functions are the sigmoid (equation 3.4) and the hyperbolic tangent (tanh, equation 3.5).

$$f\,(x) = sigmoid\,(x) = \frac{1}{1 + e^{-x}} \tag{3.4}$$

$$f\,(x) = tanh\,(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \tag{3.5}$$

For both cases, the transfer function levels off and approaches fixed limits for large negative and large positive inputs. Saturation is a feature usually forced by the application. For the filters, the percentage amount for tuning has to stay in a fixed range of [-30%, +30%] of the nominal value. The sigmoid activation function (Figure 13) provides a [0,1] range, while the hyperbolic tangent (Figure 14) provides a [-1,1] range.

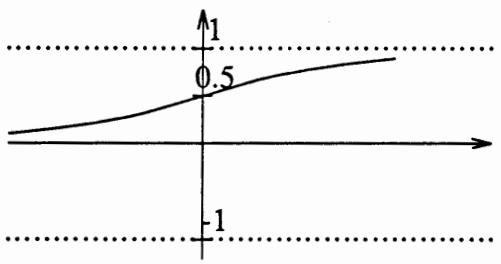


Figure 13. Sigmoid.

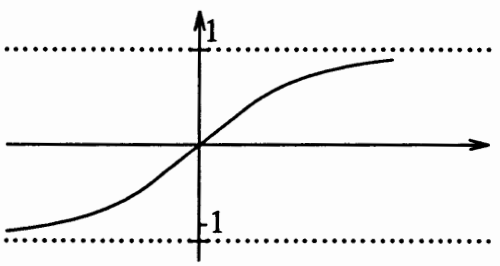


Figure 14. tanh.

III.3 THE BACKPROPAGATION TRAINING ALGORITHM

III.3.1 The idea

The backpropagation training algorithm may be summarized as follows. It provides that a notion of weight space exists. This step has already been accomplished through equations 3.1 to 3.3. The algorithm is based on a strategy called gradient descent. The task is to minimize a criterion or error function.

More detailed the algorithm includes the following steps. If $t_i^{(p)}$ are the target output values and $o_i^{(p)}$ are the actual output values of the NN for the p^{th} input pattern, then one trains the NN by minimizing the error-function given through:

$$E = \sum_{p=1}^{P} E^{(p)} = \frac{1}{2} \sum_{p=1}^{P} \sum_{i=1}^{M} (t_i^{(p)} - o_i^{(p)})^2 \tag{3.6}$$

In order to simplify the mathematics, the threshold value is assumed to equal zero. Substituting equation 3.3 in equation 3.6 results in :

$$E = \sum_{p=1}^{P} E^{(p)} = \frac{1}{2} \sum_{p=1}^{P} \sum_{i=1}^{M} (t_i^{(p)} - f(\sum_{j=1}^{L1} w_{ij} h_j^{(p)}))^2 \tag{3.7}$$

Equation 3.7 adds an extra dimension to the weight space defined through equations 3.1 to 3.3. Therefore one can think of the error-function E as a surface in weight space. The goal is to find the minimum of this surface. $E^{(p)}$ depends only on the weights and the problem patterns. The problem patterns are known and fixed. Thus only the weights can be changed to minimize the error-function. Provided we are at a certain point in weight space, it is obvious that weight changes should occur in the direction of and proportional to the negative gradient of E, with respect to the weights w_{ij}.

In general, the backpropagation training algorithm works on multilayer NNs. This makes evident that the weights of different layers have to be updated. Updating the weights of the output layer is accomplished relative easily, as equations 3.10 to 3.12 show. Updating the weights of one or more hidden layers is a more complicated task

(equations 3.13 to 3.14). There is no method to determine the correct output of a hidden PE in advance. Consequently the error in the output of a hidden PE cannot be calculated, although the actual output might be known. Otherwise it is understandable that the error in the output of a hidden PE has an influence on all the error terms on the output layer. Accordingly the error of all output PEs to which the respective hidden PE is connected, is backpropagated. Hence the name for this training algorithm is backpropagation of error. Mathematically this trick is accomplished through the calculation of the gradient of the error-function E with respect to the weights of the hidden layer (equation 3.13). After the gradient of the error-function E, with respect to the respective weight, is calculated, the weights are updated according to

$$w(t+1) = w(t) + \Delta w(t) \tag{3.8},$$

where

$$\Delta w(t) = -\eta \nabla E^{(p)} \tag{3.9}.$$

The factor η is called the learning rate. The value of η is positive and usually less than 1 [9]. The exact value will be discussed in Chapter IV.

III.3.2 Mathematical basics

The remainder of this Chapter describes the derivation of the gradient of the error-function, with respect to both, the weights of the output layer and the weights of the hidden layer. It is assumed that the NN comprises only one hidden layer.

The gradient of the error-function with respect to the weights of the output layer derives to :

$$\frac{\partial E^{(p)}}{\partial w_{ij}} = \frac{\partial E^{(p)}}{\partial o_i^{(p)}} \frac{\partial o_i^{(p)}}{\partial net_i^{(p)}} \frac{\partial net_i^{(p)}}{\partial w_{ij}} \tag{3.10}.$$

Further analysis yields:

$$\frac{\partial E^{(p)}}{\partial w_{ij}} = -\left(t_i^{(p)} - o_i^{(p)}\right)(f_i)' h_j^{(p)} = -\delta_i^{(p)} h_j^{(p)} \tag{3.11},$$

where $\delta_i^{(p)}$ is defined by:

$$\delta_i^{(p)} = (t_i^{(p)} - o_i^{(p)})(f_i)' \tag{3.12}.$$

Hereby $(f_i)'$ expresses the partial derivative of the activation function with respect to its argument. For certain activation functions f, the partial derivative can easily be calculated. Mathematical analysis for a linear activation function yields $(f_i)' = 1$, whereas for the sigmoid activation function the result is determined by $(f_i)' = f_i\,(1 - f_i\,) = o_i^{(p)}(1 - o_i^{(p)})$.

The gradient of the error-function with respect to the weights of the hidden layer derives to:

$$\frac{\partial E^{(p)}}{\partial w_{jk}} = \frac{1}{2} \sum_{i=1}^{M} \frac{\partial (t_i^{(p)} - o_i^{(p)})2}{\partial w_{jk}} \tag{3.13}.$$

Equation 3.13 may be rewritten as:

$$\frac{\partial E_{(p)}}{\partial w_{jk}} = -\sum_{i=1}^{M} (t_i^{(p)} - o_i^{(p)}) \frac{\partial o_i^{(p)}}{\partial net_i} \frac{\partial net_i}{\partial h_j} \frac{\partial h_j}{\partial net_k} \frac{\partial net_k}{\partial w_{jk}}$$

$$\frac{\partial E_{(p)}}{\partial w_{jk}} = -f'(net_k) i_k \sum_{i=1}^{M} (t_i^{(p)})\, f'(net_i)\, w_{ij}$$

$$\frac{\partial E_{(p)}}{\partial w_{jk}} = -f'(net_k) isubk \sum_{i=1}^{M} \delta_i^{(p)} w_{ij} \tag{3.14}$$

The latter form of equation 3.14 makes evident that every update on the hidden layer depends on all error terms $\delta_i^{(p)}$ on the output layer. As already suggested through the form of equation 3.6, weight updates are performed as each training pattern is processed. It would be possible to sum the error of all patterns and then make one update to the weights. The minimazation process would then be described by $E = \sum_{p=1}^{P} E^{(p)}$. Yet, it has little advantage and requires storing a large amount of data [9].

III.4 NEURAL COMPUTING

The question which still has to be answered is, how will the NN be realized? Although hardware implementations [17,18,19] exist, most current NNs are simulated by sequential computers. Since the goal of this paper is to investigate the efforts required to tune second-order, integrated filters using NNs, an actual hardware implementation is not required. A simulation of NNs is sufficient. The NeuralWorks Professional II/Plus software package from NeuralWare incorporation was known to the author before. It is available on the Unix system of Portland State University and includes all the features discussed. Consequently it has been used for simulation of the NN.

To understand what is going on, while using the software, it seems important to the author to keep some characteristics in mind. All PEs in a layer fire synchronously and each layer fires sequentially. The updating of the weights is accomplished similarly. This means all weights (in each layer) are updated together and at one step in time. Whether the weights are updated after each pattern has been presented, depends on how the network parameters are set up in detail. The approach taken in this paper sets the respective parameters (Epoch) to the value one, such that the weights are indeed incrementally updated. This approach led to good results.

CHAPTER IV

NEURAL NETWORK APPROACH FOR THE PRESENT PROBLEM

IV.1 DATA GENERATION

Starting work with NNs requires a set of input-output patterns (example data pairs), which represent the desired problem domain. From the problem definition previously given, it is already known that the input to the filter are the 49 differences of the actual and the desired magnitude of the respective filter's (Circuit A or Circuit B) frequency response. It would have been possible to present the actual magnitude of the transfer function to the NN (rather than the error values). As discussed in data preparation for a NN [20], changes in amount are preferred to absolute values, as they provide a smaller range, and consequently small-value differences are more meaningful to the NN.

The sampling of the filter at 49 frequency components was performed to represent the magnitude of the transfer function as a whole pattern. Accordingly, all 49 frequency samples have to be presented to the NN at once. This requires 49 PEs in the input layer of the NN, resulting in the fact that each PE recognizes changes in the magnitude for a fixed value of the frequency.

The data were created using the mathematical software package called Matlab (see APPENDIX B). The frequency samples were taken for a range from 0.2 to 2.6 times the nominal pole frequency (1 MHz for both Circuit A and Circuit B) and with a stepsize of 0.05 times the nominal pole frequency. The components of the filter are varied between -30% and +30% of their nominal value. For both cases, training data generation as well as testing data generation, each component was incrementally changed with a step-size of 5%. The training data vary between -30% and +30%, whereas the testing data take on

the in-between values in the range from -27.5% to +27.5%. For the data generation of Circuit B, the constraint that C_1/C_2 stays constant within 0.1% of its nominal value has been used. C_1 and C_2 differ at most by 5%, in either direction.

For each nominal quality factor Q_{p0}, a training set and a testing set has been created. The above changes result in the fact that the NN for Circuit A has been trained on 2197 different transfer functions, whereas the NN for Circuit B has been trained on 6253 different transfer functions. The testing set consists of 1728 (Circuit A) or 4896 (Circuit B) records.

Computing the input-output patterns is clear cut for Circuit A. One just has to keep track of the Δs which change the transfer function and store them as the desired outputs. The transfer function of a low-pass filter with unity gain is determined by only two parameters. To enable the NN to tune three parameters, additional information is required to avoid a non-unique inverse mapping. For the dc case, the capacitor C and the inductor L of Circuit A can be neglected. Hence using a resistor R_0 (see Figure 15), whose value is exactly known, the value of R can be determined.

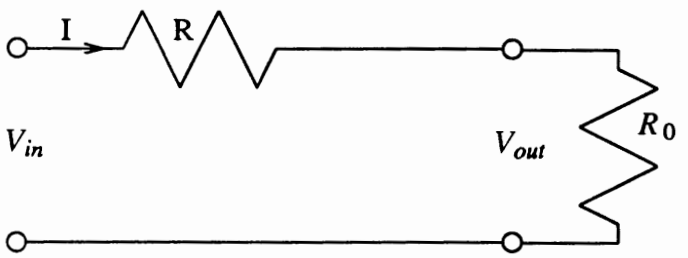


Figure 15. Measurement of resistor R.

Viewing Figure 15 makes evident that:

$$\frac{V_{out}}{V_{in}} = \frac{R_0}{R + R_0}$$

This can be rewritten as:

$$R = R_0\left(\frac{V_{in}}{V_{out}} - 1\right),$$

where V_{in} and V_{out} have to be measured. The additional information is provided through this measurement and is included as the fiftieth input of the NN.

For Circuit B we have to go through a mathematical analysis to prove that the g_m s can be tuned, such that they cancel out the error caused through the capacitors. After tuning the transconductances, the components are described by:

$$g_{m1} = g_{m10}\left(1 + \frac{\Delta g_{m1a} + \Delta g_{m1t}}{g_{m10}}\right) \tag{4.1a}$$

$$g_{m2} = g_{m20}\left(1 + \frac{\Delta g_{m2a} + \Delta g_{m2t}}{g_{m20}}\right) \tag{4.1b}$$

$$C_1 = C_{10}\left(1 + \frac{\Delta_{1a}}{C_{10}}\right) \tag{4.2a}$$

$$C_2 = C_{20}\left(1 + \frac{\Delta_{2a}}{C_{20}}\right) \tag{4.2b}$$

where the subscript 0 denotes the nominal value. The Δ_a-term describes the actual amount the respective component is off. The Δ_t-term, which occurs only in equations (4.1), stands for the amount the respective transconductance has been tuned. Substituting equations (4.1) and (4.2) in equations (2.9) and (2.10) results in the actual pole frequency ω_a and the actual quality factor Q_a.

$$\omega_a = \sqrt{\frac{g_{m10}g_{m20}}{C_{10}C_{20}}}\;\frac{\sqrt{\left(1 + \frac{\Delta g_{m1a} + \Delta g_{m1t}}{g_{m10}}\right)\left(1 + \frac{\Delta g_{m2a} + \Delta g_{m2t}}{g_{m20}}\right)}}{\sqrt{\left(1 + \frac{\Delta C_{1a}}{C_{10}}\right)\left(1 + \frac{\Delta C_{2a}}{C_{20}}\right)}} \tag{4.3}$$

$$Q_a = \sqrt{\frac{g_{m10}C_{10}}{g_{m20}C_{20}}}\;\frac{\sqrt{\left(1 + \frac{\Delta g_{m1a} + \Delta g_{m1t}}{g_{m10}}\right)\left(1 + \frac{\Delta C_{1a}}{C_{10}}\right)}}{\sqrt{\left(1 + \frac{\Delta g_{m2a} + \Delta g_{m2t}}{g_{m20}}\right)\left(1 + \frac{\Delta C_{2a}}{C_{20}}\right)}} \tag{4.4}$$

The nominal pole frequency ω_{p0} and the nominal quality factor Q_{p0} are defined by:

$$\omega_{p0} = \sqrt{\frac{g_{m10}g_{m20}}{C_{10}C_{20}}} \tag{4.5}$$

$$Q_{p0} = \sqrt{\frac{g_{m10}C_{10}}{g_{m20}C_{20}}} \tag{4.6}$$

It is obvious that after tuning, the actual and the nominal values of the pole frequency should equal each other. The analogous statement is true for the quality factor. Consequently the second term in equation (4.3) and the second term in equation (4.4) have to equal 1.

$$\frac{\sqrt{(1+\frac{\Delta g_{m1a}+\Delta g_{m1t}}{g_{m10}})(1+\frac{\Delta g_{m2a}+\Delta g_{m2t}}{g_{m20}})}}{\sqrt{(1+\frac{\Delta C_{1a}}{C_{10}})(1+\frac{\Delta C_{2a}}{C_{20}})}} \equiv 1 \tag{4.7}$$

$$\frac{\sqrt{(1+\frac{\Delta g_{m1a}+\Delta g_{m1t}}{g_{m10}})(1+\frac{\Delta C_{1a}}{C_{10}})}}{\sqrt{(1+\frac{\Delta g_{m2a}+\Delta g_{m2t}}{g_{m20}})(1+\frac{\Delta C_{2a}}{C_{20}})}} \equiv 1 \tag{4.8}$$

Further mathematical analysis yields :

$$\frac{\Delta g_{m1t}}{g_{m10}} = \frac{\Delta C_{2a}}{C_{20}} - \frac{\Delta g_{m1a}}{g_{m10}} \tag{4.9}$$

$$\frac{\Delta g_{m2t}}{g_{m20}} = \frac{\Delta C_{1a}}{C_{10}} - \frac{\Delta g_{m2a}}{g_{m20}} \tag{4.10},$$

in order to satisfy the constraints given through equations (4.7) and (4.8). Accordingly equations (4.9) and (4.10) define the output of the NN.

IV.2 EVALUATION OF THE NEURAL NETWORK OUTPUT

Evaluation of the NNs output at this point might seem to be a step ahead. Yet this is not the case. When setting up NN experiments, one always has to think about the evaluation in advance. How can it be judged, if or how well the NN does learn? A cri-

terion is needed to determine the performance of the NN. The meaning of the RMS-error (Root-Mean-Squared-error), provided by the software, is very fuzzy. It shows whether there is a tendency which is in general good or bad. A more precise criterion is desirable.

The evaluation of Circuit A is accomplished utilizing a c-file (see APPENDIX C.1), which calculates both, the maximum error of each component (in either direction) and the Euclidean distance (average, maximum, minimum) of the actual output of the NN with respect to the desired and calculated output. This criterion enables us to judge if the NN is able to determine the changes in the parameters.

The Matlab-evaluation-file for Circuit B (see APPENDIX C.3) even accomplishes a more subtle task. It is supposed that the NN does not deliver exact values for $\Delta g_{m1t}/g_{m10}$ and $\Delta g_{m2t}/g_{m20}$ (equations 4.9 and 4.10). Defining the error-terms e_1 and e_2 allows us to calculate the actual values of the normalized pole frequency ω_a/ω_{p0} and the normalized quality factor Q_a/Q_{p0}.

$$e_1 = \frac{\Delta g_{m1NN}}{g_{m10}} - \frac{\Delta g_{m1t}}{g_{m10}} \tag{4.11}$$

$$e_2 = \frac{\Delta g_{m2NN}}{g_{m20}} - \frac{\Delta g_{m2t}}{g_{m20}} \tag{4.12}$$

where the subscript t denotes the calculated tuning amount. NN stands for the actual output of the NN.

$$\frac{\omega_a}{\omega_{p0}} = \sqrt{1 + \frac{e_1}{A} + \frac{e_2}{B} + \frac{e_1 e_2}{AB}} \tag{4.13}$$

$$\frac{Q_a}{Q_{p0}} = \frac{\sqrt{1 + \frac{e_1}{A}}}{\sqrt{1 + \frac{e_2}{B}}} \tag{4.14},$$

where A and B are defined by:

$$A = 1 + \frac{\Delta C_{2a}}{C_{20}}, B = 1 + \frac{\Delta C_{1a}}{C_{10}}.$$

More evident, the latter paragraph may be summarized by stating that the determination of ω_a/ω_{p0} and Q_a/Q_{p0} requires keeping track of two steps, the calculation of the

error-terms e_1 and e_2 and the calcualtion of the term A and B. The latter terms are calculated simultaneously with data generation (see APPENDIX B). The error-terms are calculated via the NN result files, the .nnr-files (see APPENDIX C.3).

Using these mathematics, the actual ω_a/ω_{p0} and Q_a/Q_{p0} as well as their maximum and minimum values are calculated. Additionally it is calculated how many records stay in a 5%- and in an 1%-error-box around the nominal value.

Finally equations (4.3) and (4.4) are used to determine the worst cases of ω_a/ω_{p0} and Q_a/Q_{p0} before tuning.

$$Maximum: \frac{\omega_a}{\omega_{p0}} = \sqrt{\frac{1.3}{0.7}\frac{1.3}{0.7}} = 1.857$$

$$Minimum: \frac{\omega_a}{\omega_{p0}} = \sqrt{\frac{0.7}{1.3}\frac{0.7}{1.3}} = 0.539$$

$$Maximum: \frac{Q_a}{Q_{p0}} = \sqrt{\frac{1.3}{0.7}\frac{0.75}{0.7}} = 1.411$$

$$Minimum: \frac{Q_a}{Q_{p0}} = \sqrt{\frac{0.7}{1.3}\frac{0.7}{0.75}} = 0.709$$

IV.3 NEURAL NETWORK EXPERIMENTS

IV.3.1 Network architecture used

Consistent throughout all experiments, the type of NN used was a Backpropagation Network with one hidden layer. The input layer consisted of 49 PEs, with the exception of the NN for Circuit A which required a fiftieth input for the measurement of R. The number of output PEs equals the number of filter components, which are to be tuned -- 3 for Circuit A and 2 for Circuit B. Eighteen hidden PEs proved to do a good job for each different nominal quality factor (TABLE I and TABLE II). Therefore a schematic of this type of NN is shown in Figure 16. Changes in the number of the hidden PEs are mentioned for the respective case.

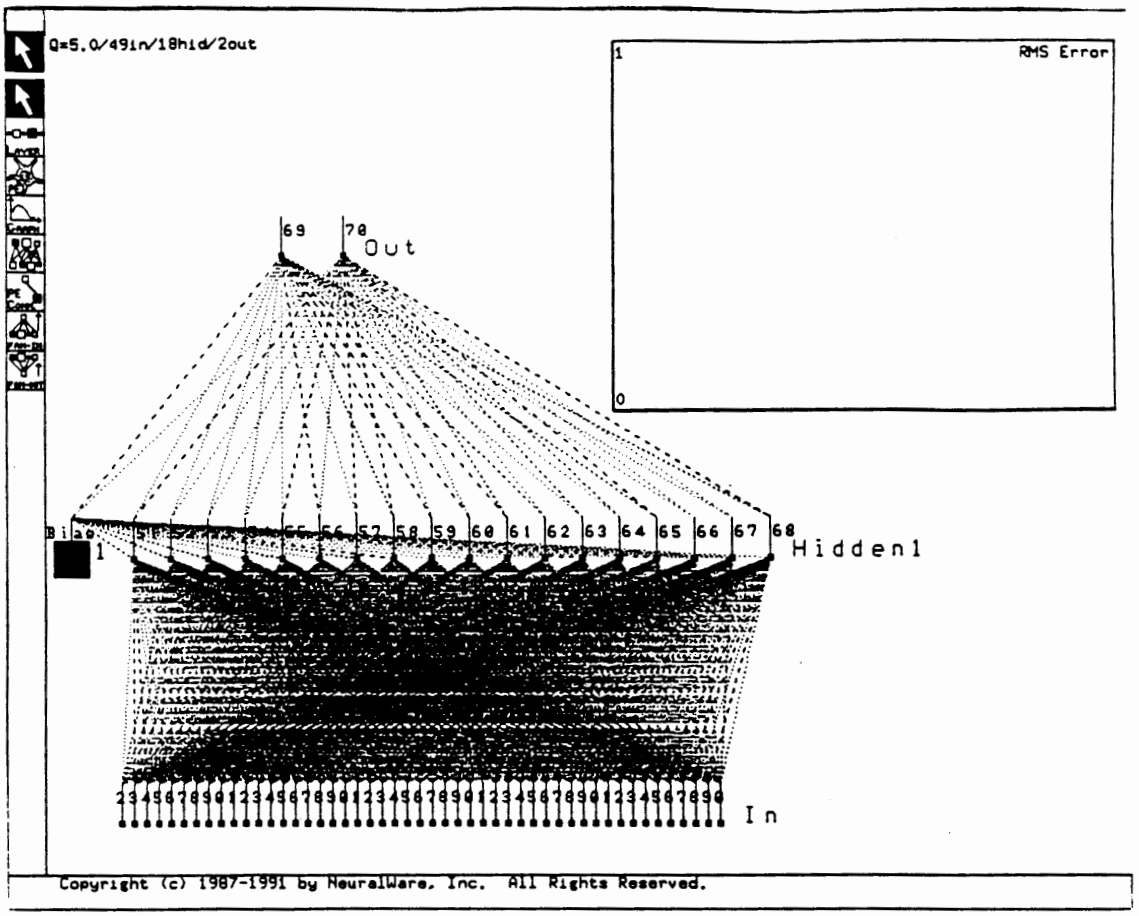


Figure 16. Backpropagation Network for the present problem.

IV.3.2 Activation function

The output of the NN takes on both negative and positive values. Accordingly, an activation function has to be chosen that takes on positive and negative values. This is obvious for the output layer. Thus the activation for all output PEs was selected to be the hyperbolic tangent.

Initial experiments with different activation functions for the hidden PEs made evident that the activation function of the hidden PEs also has to be the hyperbolic tangent, in order to get reasonable outputs.

IV.3.3 Network dynamics and learning rate

An important question, while working with NNs, is how fast does the NN learn. Asking this question already implies that the NN converges to an equilibrium point, which results in at least very small errors of the NN output. For the present problem, convergence is guaranteed (CHAPTER V RESULTS) within a relatively small number of iterations of about 200,000 to 300,000 , depending on the nominal quality factor. Remember that the NN is trained on 6253 records. Accordingly, the NN sees each record only 30 to 50 times until the weights have been adjusted.

Convergence dynamics is coupled with the value of the learning rate. Beginning experiments started with high learning rates close to the value 1. Lowering this value to 0.5 did not make any changes. In both cases the NN could not solve the problem of encoding the input-output relationship. Talking in terms of the weight space notion, it seems that the high learning rate resulted in an inability of the NN to find an equilibrium point. Obviously, the error-function (equation 3.6) could not be minimized, rather it jumped over the minimum. Consequently the learning rate was decreased to values of 0.3 for the first 10000 iterations and down to about 0.001 for the last iterations (see APPENDIX D).

IV.3.4 Testing the Neural Network

After a NN has been trained, it has to be tested. Two possibilities come up to accomplish this step. On the one hand one may test the NN on the data presented during training. Applying nothing but these previously seen data to the NN can only answer questions regarding network dynamics and memorization of data. Since the task of the NN is to tune a real-world filter, it can never be guaranteed that the NN is trained with all parameter combinations. Thus a task demanded from the NN is to perform a good generalization on unseen data. The meaning of good is defined by the evaluation criterion (Chapter IV.2).

In this context the number of the weights has to be addressed. A rule of thumb provides that the number of training records should be 5 to 10 times as large as the number of weights, in order to enable a good generalization of the NN. Even if this rule is obeyed, a good generalization can never be guaranteed. The data available for the experiments yield values of 2.3 (for Circuit A's NN) and 6.8 (for Circuit B's NN) for the ratio of the number of training records and the number of weights. If the number of hidden PEs is too large, or if the number of training records is too small, the NN might tend to memorize the data. In such a case, the response on unseen data would be poor [20].

CHAPTER V

RESULTS FOR CIRCUIT B

Results of the NN approach to the present problem are, to a large extent, given through graphic presentations. This choice has been made, because it is more apparent than written text. In order to enable a more in-depth investigation of the results, the results are summarized in Tables.

V.1 LEARNING DYNAMICS

The following plots (Figures 17. to 25.) show the learning dynamics for both training and generalization. A solid line represents the results of the previously seen records. A dotted line shows the outcome of the NN when presented with unseen data. The upper lines indicate the percentage of records included in a 5% error-box around the nominal values of the quality factor and of the pole frequency. The lower lines indicate the percentage of records which stay in a 1% error-box around the nominal values of the quality factor and of the pole frequency.

Each of these plots shows that the learning process of the NN is relatively fast. After 200,000 to 300,000 iterations, the NN already achieves its best performance. Further training might still change the weights. Despite this, the performance is not normally improved.

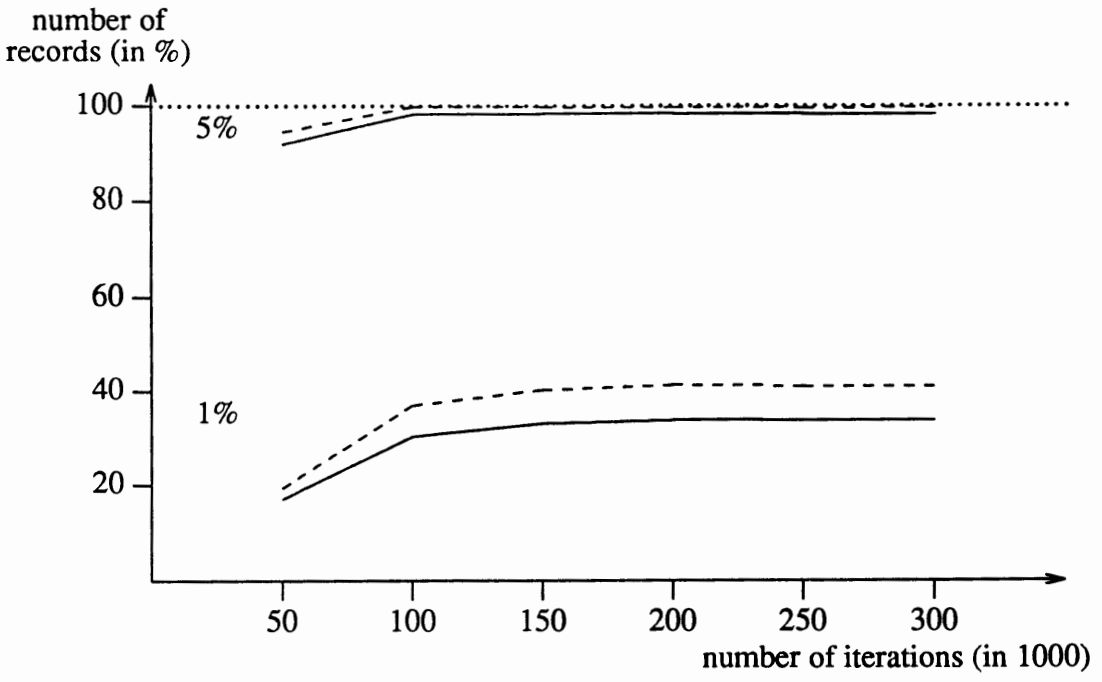


Figure 17. Network dynamics for $Q_{p0} = 5.0$,
($\omega_{p0} = 10 Mrad/s$ in Figures 17. to 25.).

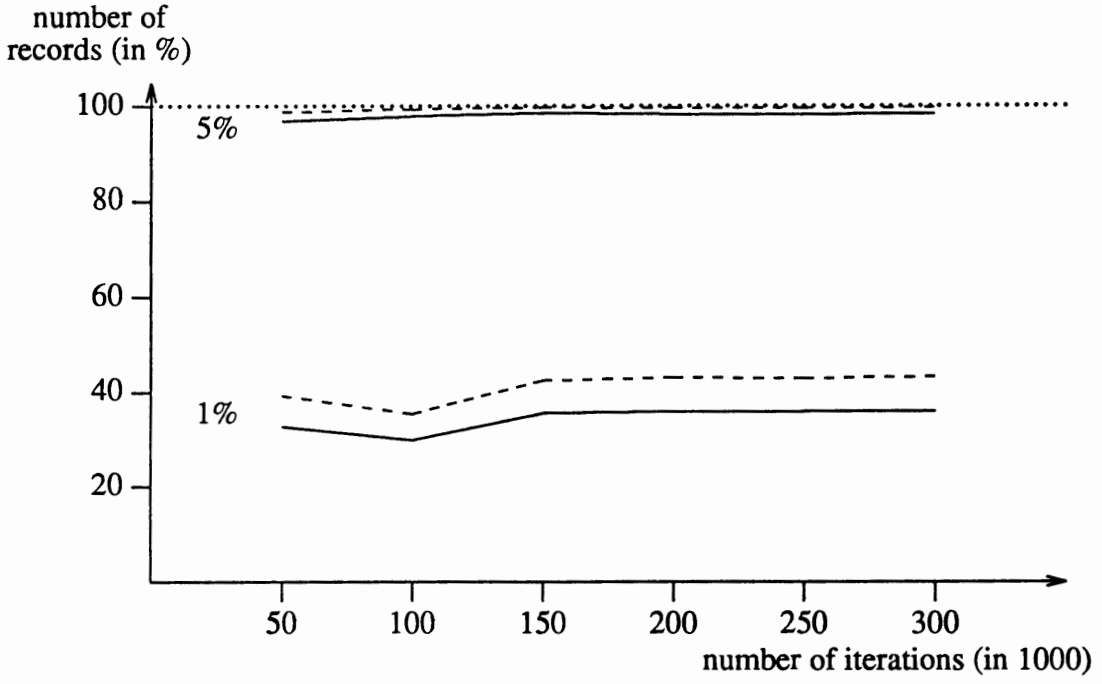


Figure 18. Network dynamics for $Q_{p0} = 3.33$,
($\omega_{p0} = 10 Mrad/s$ in Figures 17. to 25.).

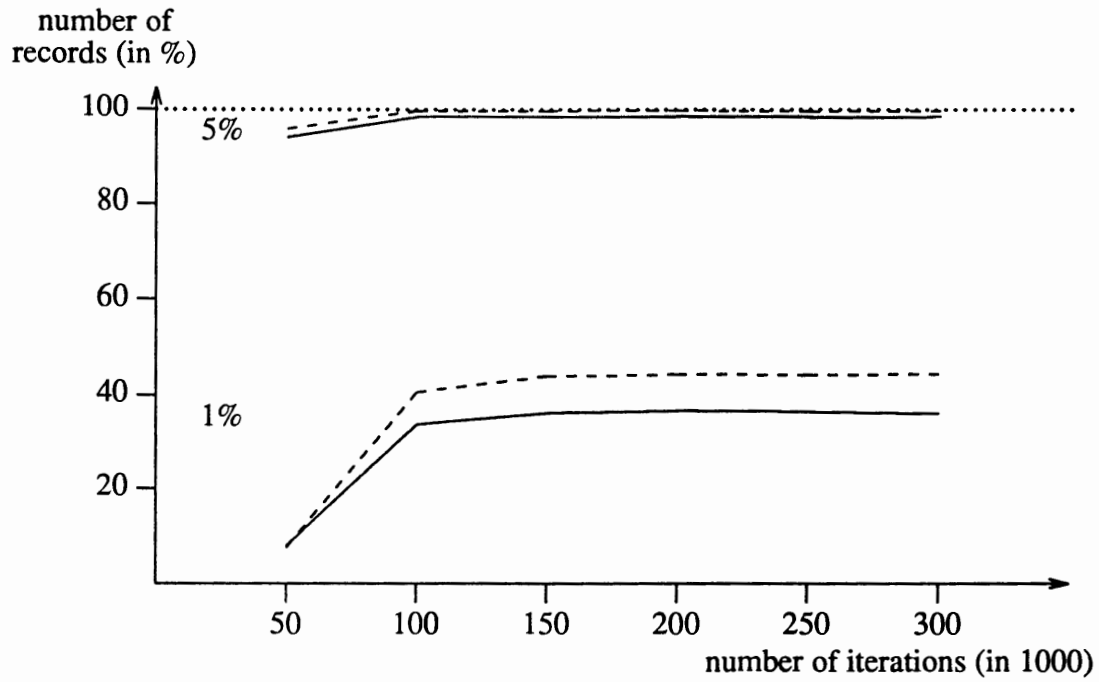


Figure 19. Network dynamics for $Q_{p0} = 2.5$,
($\omega_{p0} = 10 Mrad/s$ in Figures 17. to 25.).

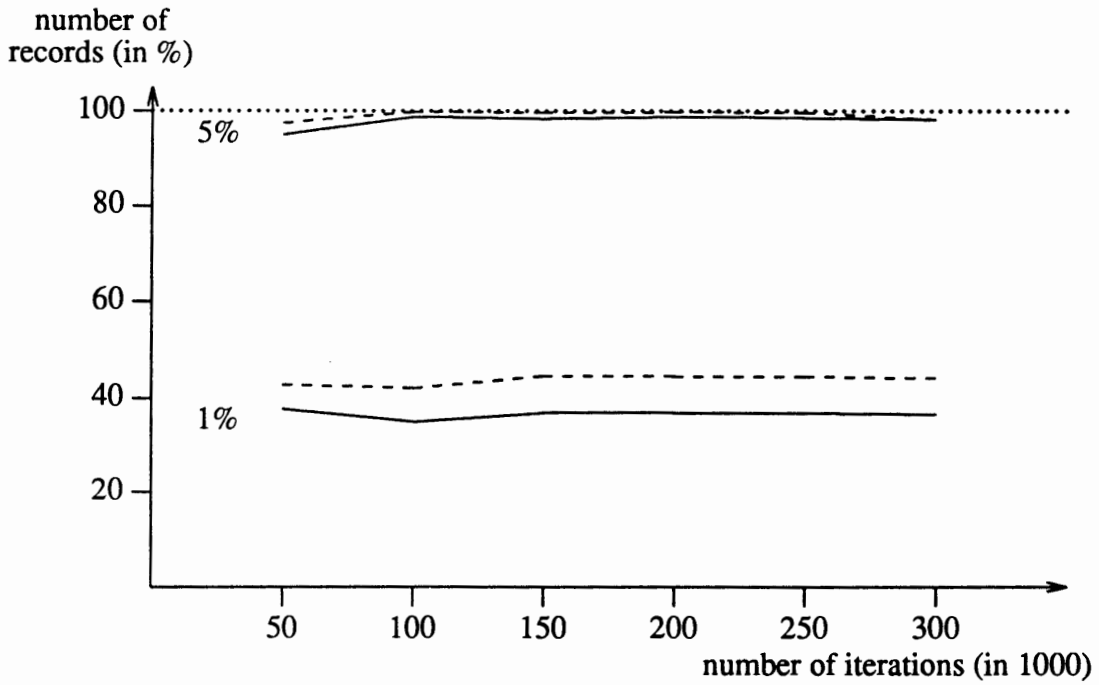


Figure 20. Network dynamics for $Q_{p0} = 1.67$,
($\omega_{p0} = 10 Mrad/s$ in Figures 17. to 25.).

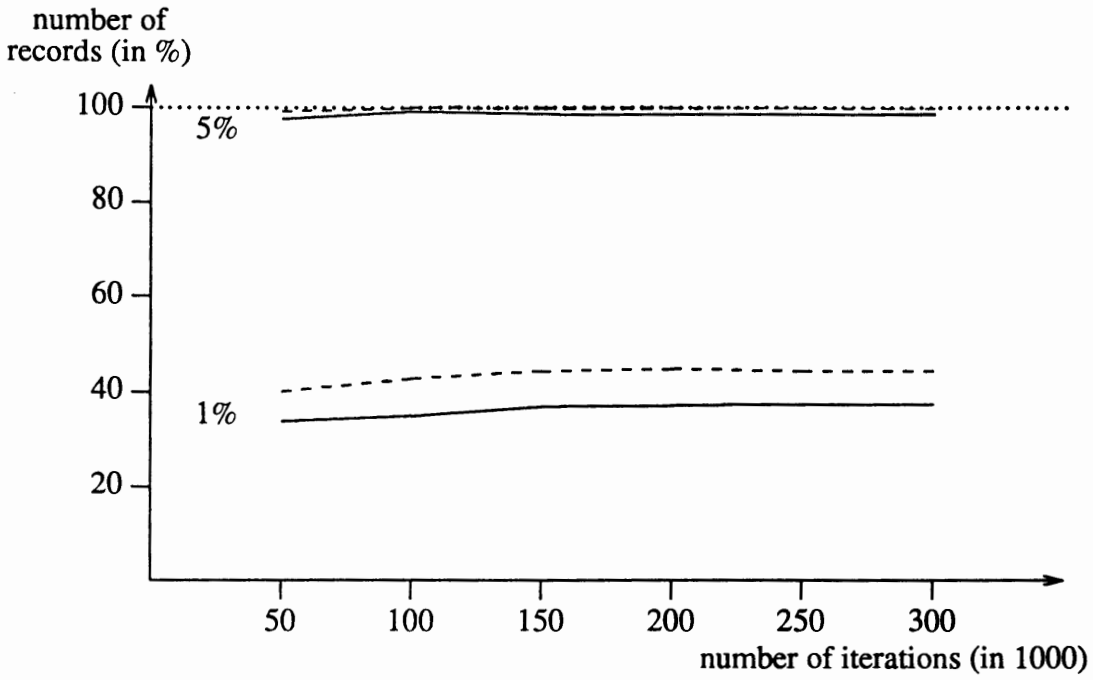


Figure 21. Network dynamics for $Q_{p0} = 1.25$, ($\omega_{p0} = 10 Mrad/s$ in Figures 17. to 25.).

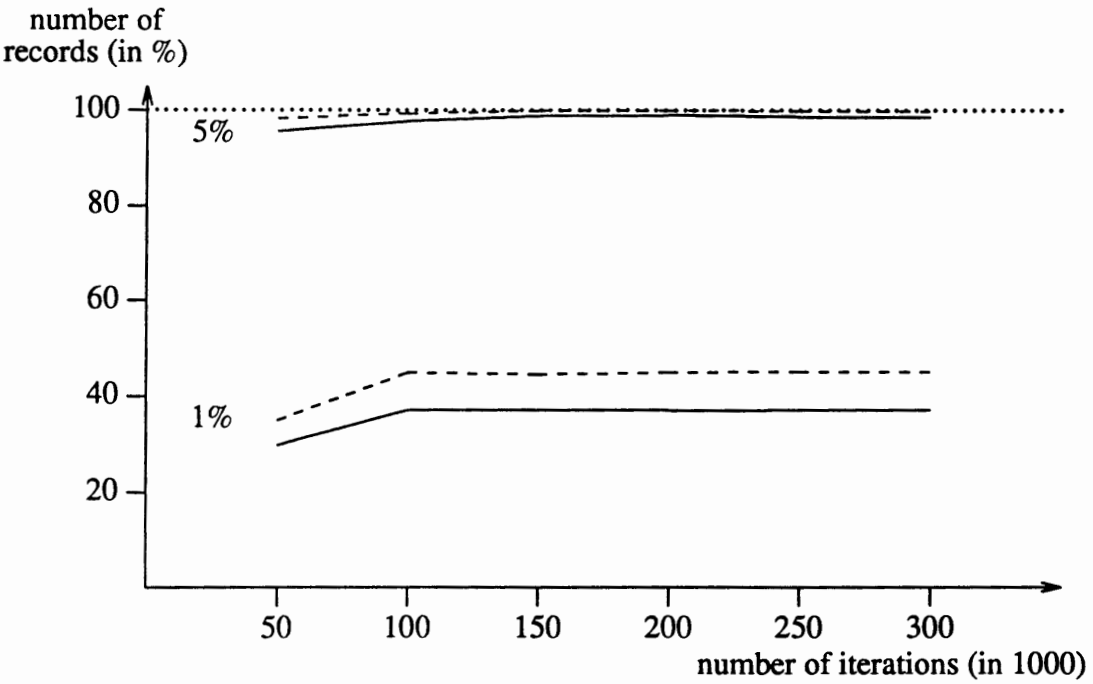


Figure 22. Network dynamics for $Q_{p0} = 1.0$, ($\omega_{p0} = 10 Mrad/s$ in Figures 17. to 25.).

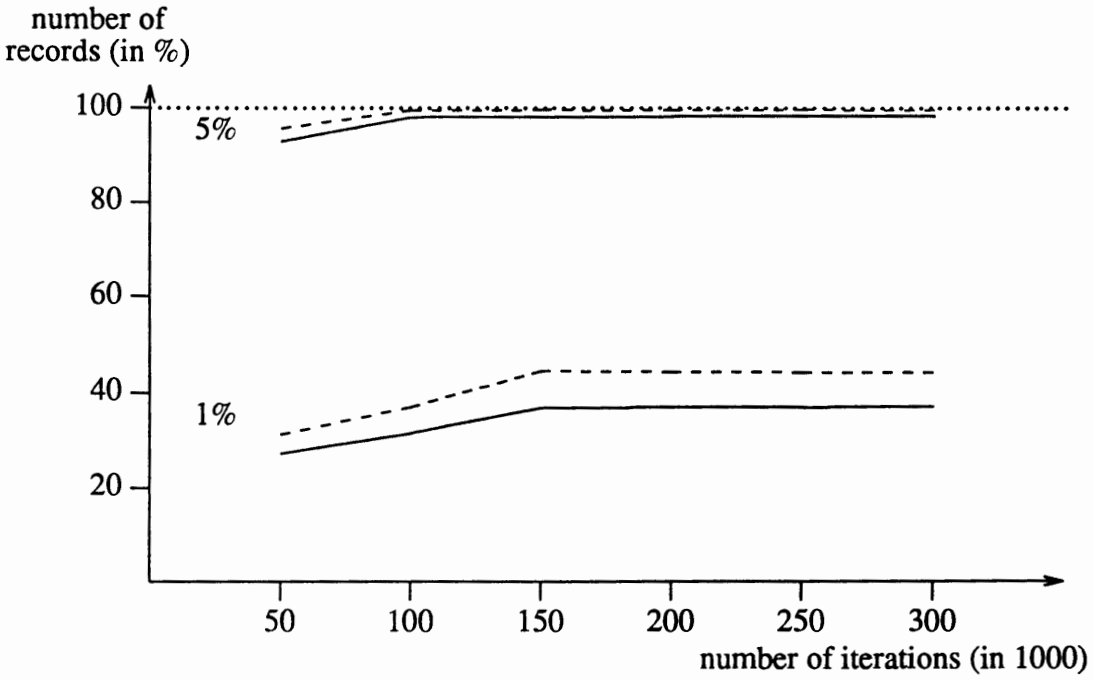


Figure 23. Network dynamics for $Q_{p0} = 0.83$, ($\omega_{p0} = 10 Mrad/s$ in Figures 17. to 25.).

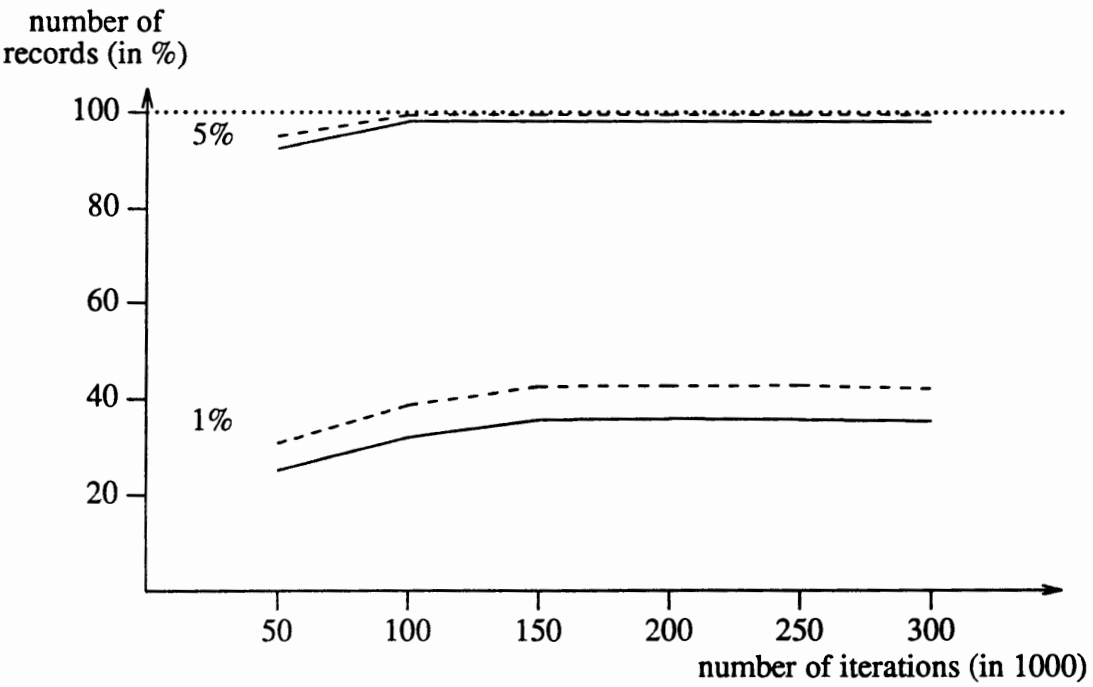


Figure 24. Network dynamics for $Q_{p0} = 0.707$, ($\omega_{p0} = 10 Mrad/s$ in Figures 17. to 25.).

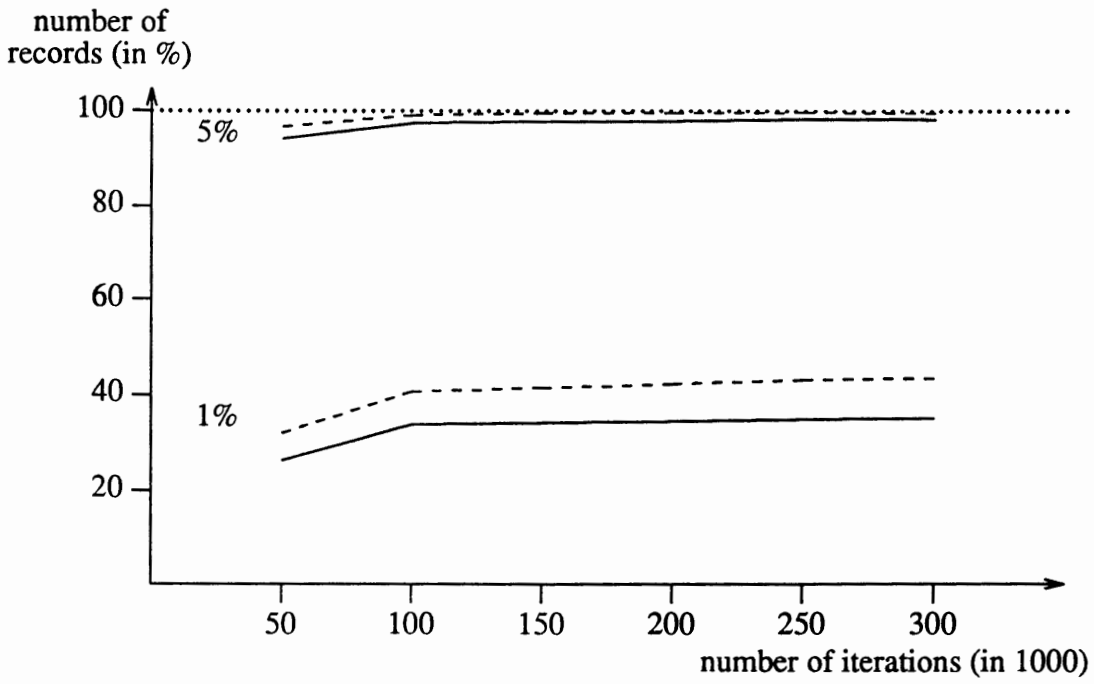


Figure 25. Network dynamics for $Q_{p0} = 0.625$, ($\omega_{p0} = 10 Mrad/s$ in Figures 17. to 25.).

V.2 PERFORMANCE

A crucial step, which has to be accomplished, in order to evaluate a NN, is the determination of the NN's performance. For the present problem, a file written in Matlab-code (Appendix C.3) calculates the actual, normalized pole frequency ω_a/ω_{p0} and the actual, normalized quality factor Q_a/Q_{p0}. 6253 training records and 4896 generalization records are too complex to be realized by the human brain at once. Yet a two-dimensional plot of the actual values (see following figures), allows us to recognize the data as clusters. This enables the observer to judge the overall performance of the NN. The significance of a single record is not neglected, in that extreme values will always be apart from the clusters.

V.2.1 Testing on 6253 previously seen records (training data)

Figure 26 shows the initial errors of the quality factor and of the pole frequency. Figures 27 and A.1 to A.9 show the response of the NN, when presented with the previously seen training records. Realizing that the scale changes from Figure 27 to Figures A.1 to A.9 reveals that the performance of the NN is accurate. The response of the NN with initial scaling is plotted only once. This choice has been made, because the plots of different nominal quality factors are almost identical for this scale.

V.2.2 Testing on 4896 previously not seen records (generalization data)

Figures 28,29, and A.10 to A.18 show the response of the NN, when presented with records not seen during training. Again the comments stated in V.2.1 have to be kept in mind. To make the results of the NN's generalization more evident, 11 samples have been selected, as shown in Figure 30. The criterion of choice of the samples was to choose records which took on either extreme initial value or extreme value after tuning. The results are shown in Figures 31,32 and A.19 to A.34.

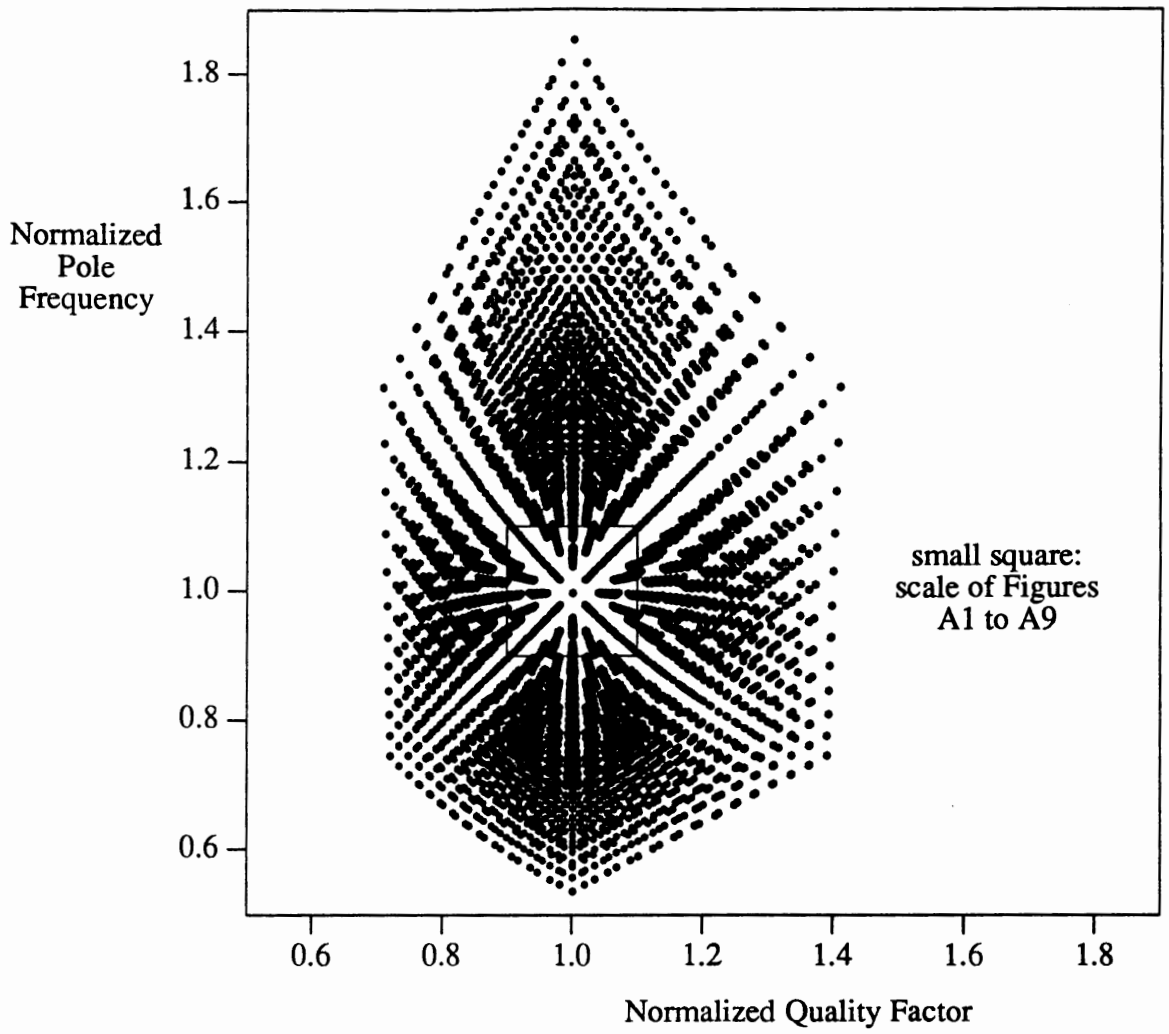


Figure 26. Initial errors of the quality factor Q_p and the pole frequency ω_p (training data).

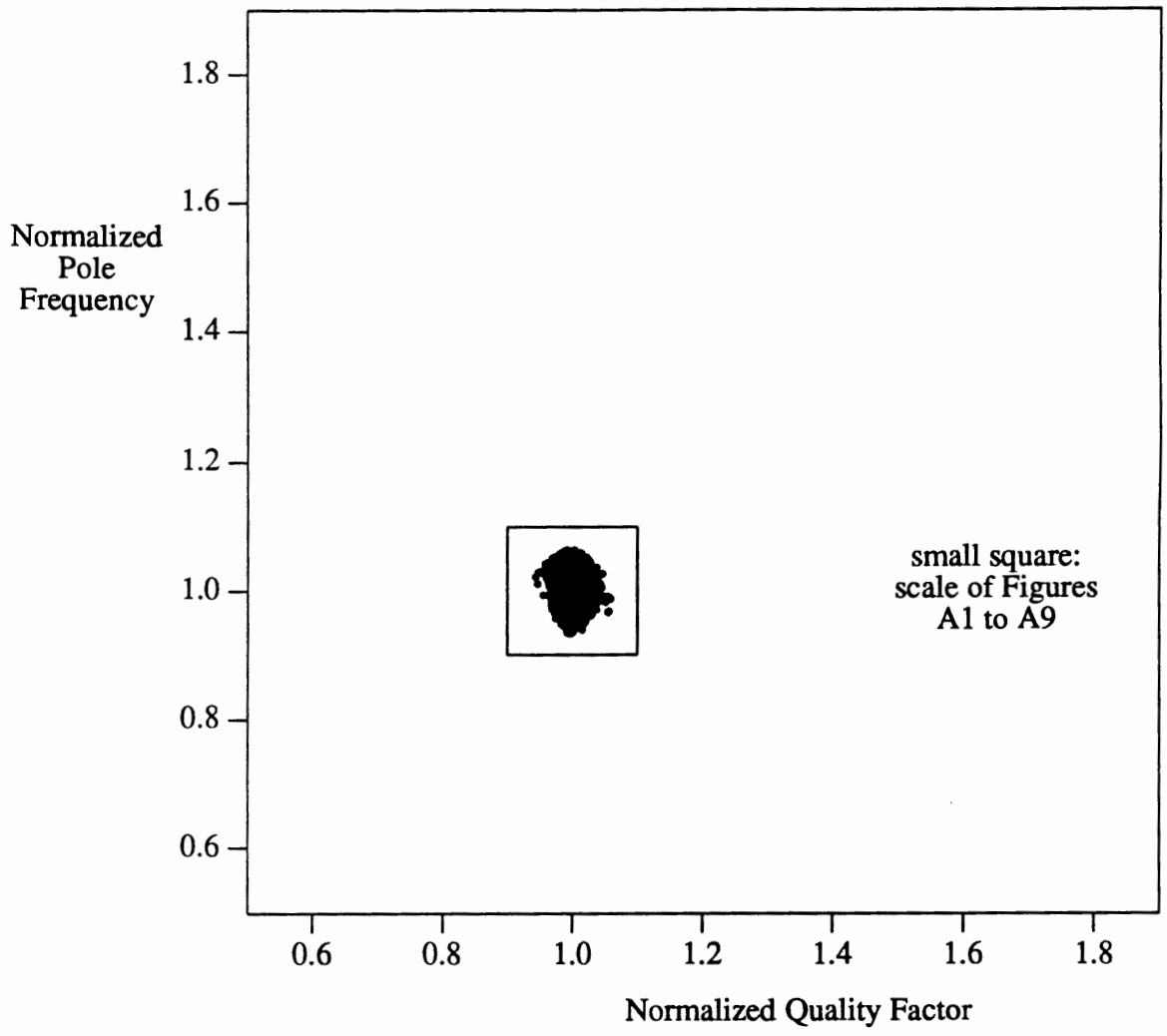


Figure 27. Residual errors of the quality factor Q_p and the pole frequency ω_p after tuning via NN (testing with training data, $Q_{p0} = 5.0$)(compare with Figure 26.).

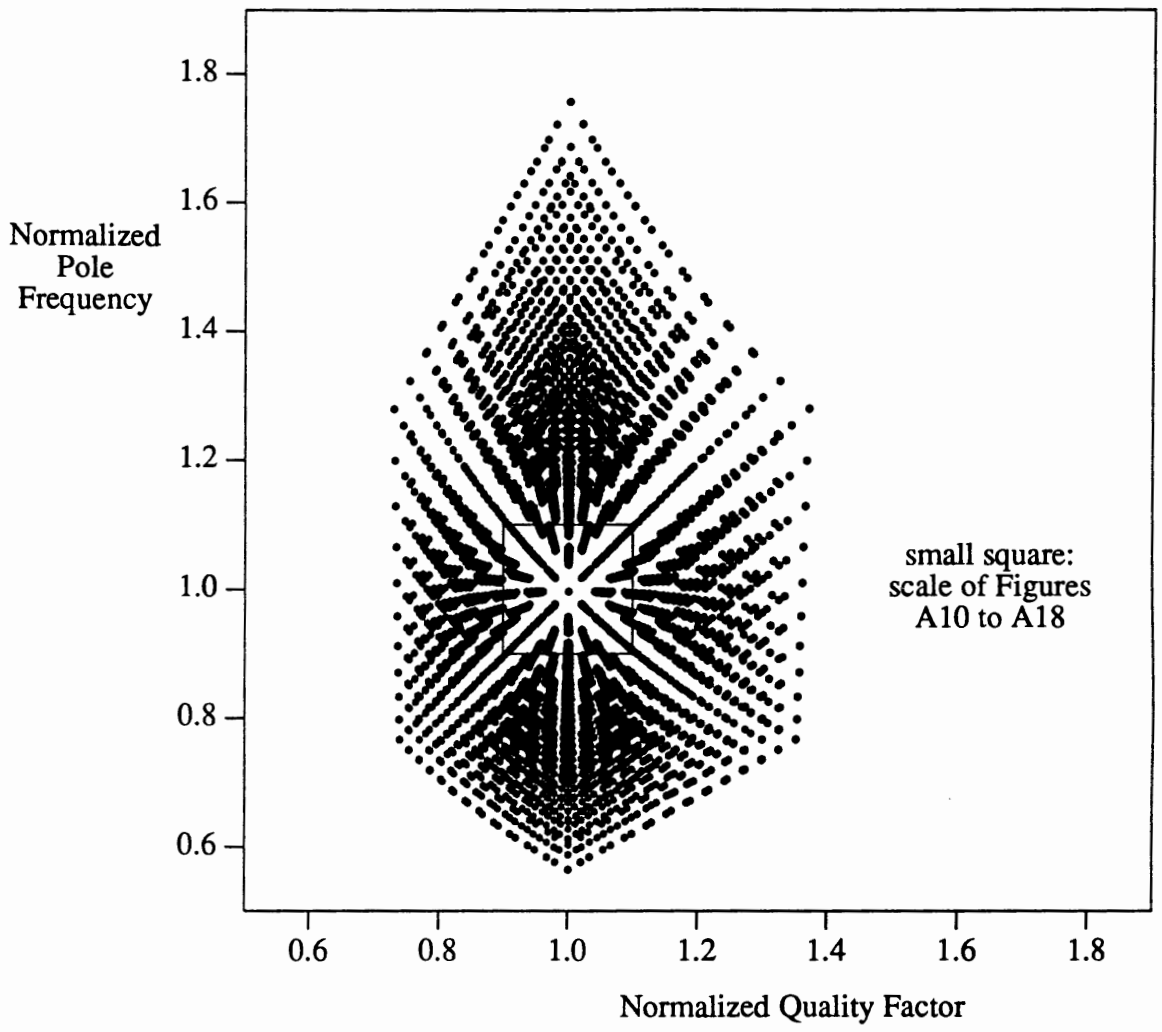


Figure 28. Initial errors of the quality factor Q_p and the pole frequency ω_p (generalization data).

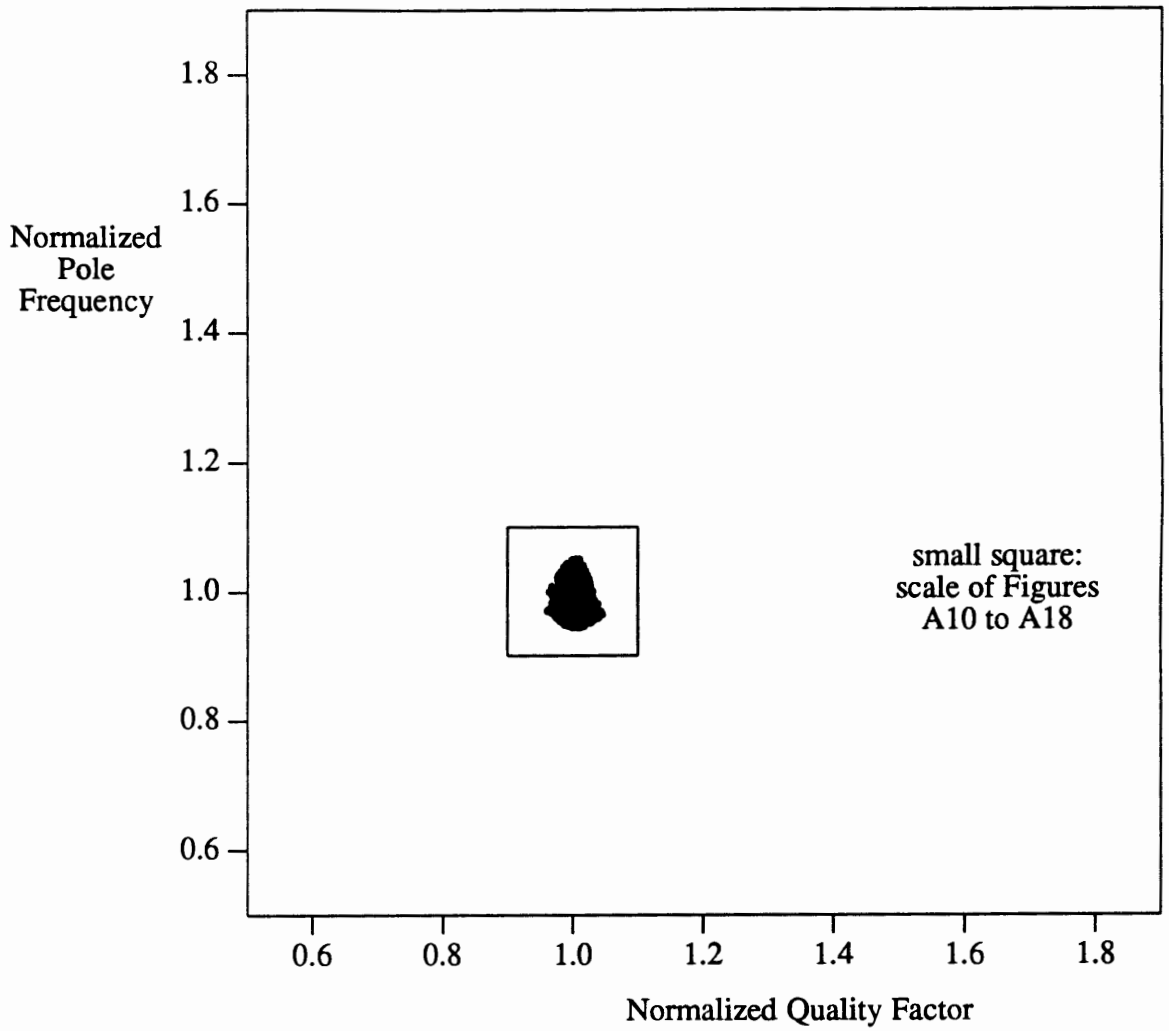


Figure 29. Residual errors of the quality factor Q_p and the pole frequency ω_p after tuning via NN (testing with generalization data, $Q_{p0} = 5.0$) (compare with Figure 28.).

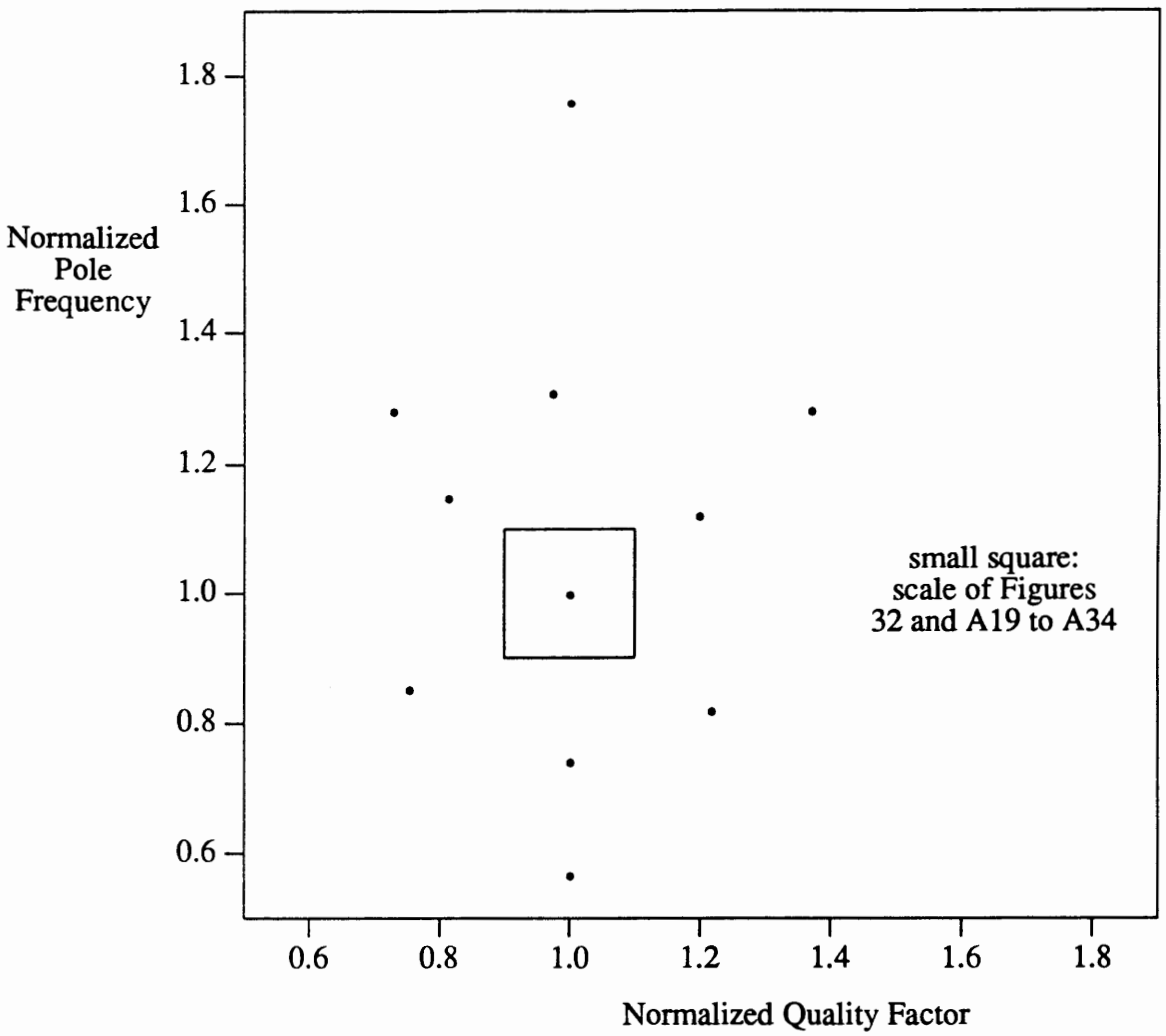


Figure 30. Selected initial errors of the quality factor Q_p and the pole frequency ω_p (11 samples, generalization data).

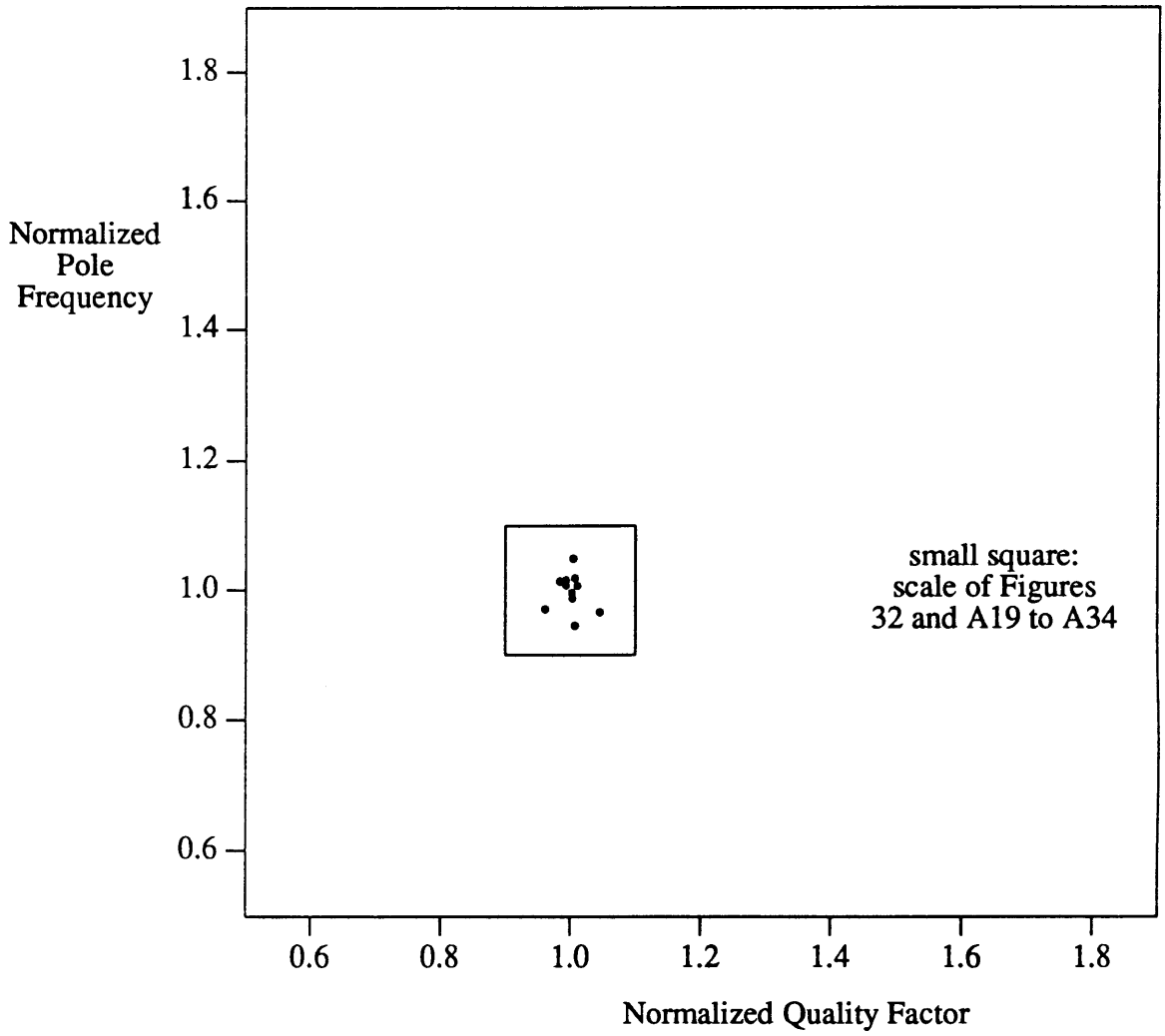


Figure 31. Residual errors of the quality factor Q_p and the pole frequency ω_p after tuning via NN (testing with 11 samples, generalization data, $Q_{p0} = 5.0$)(compare with Figure 30.).

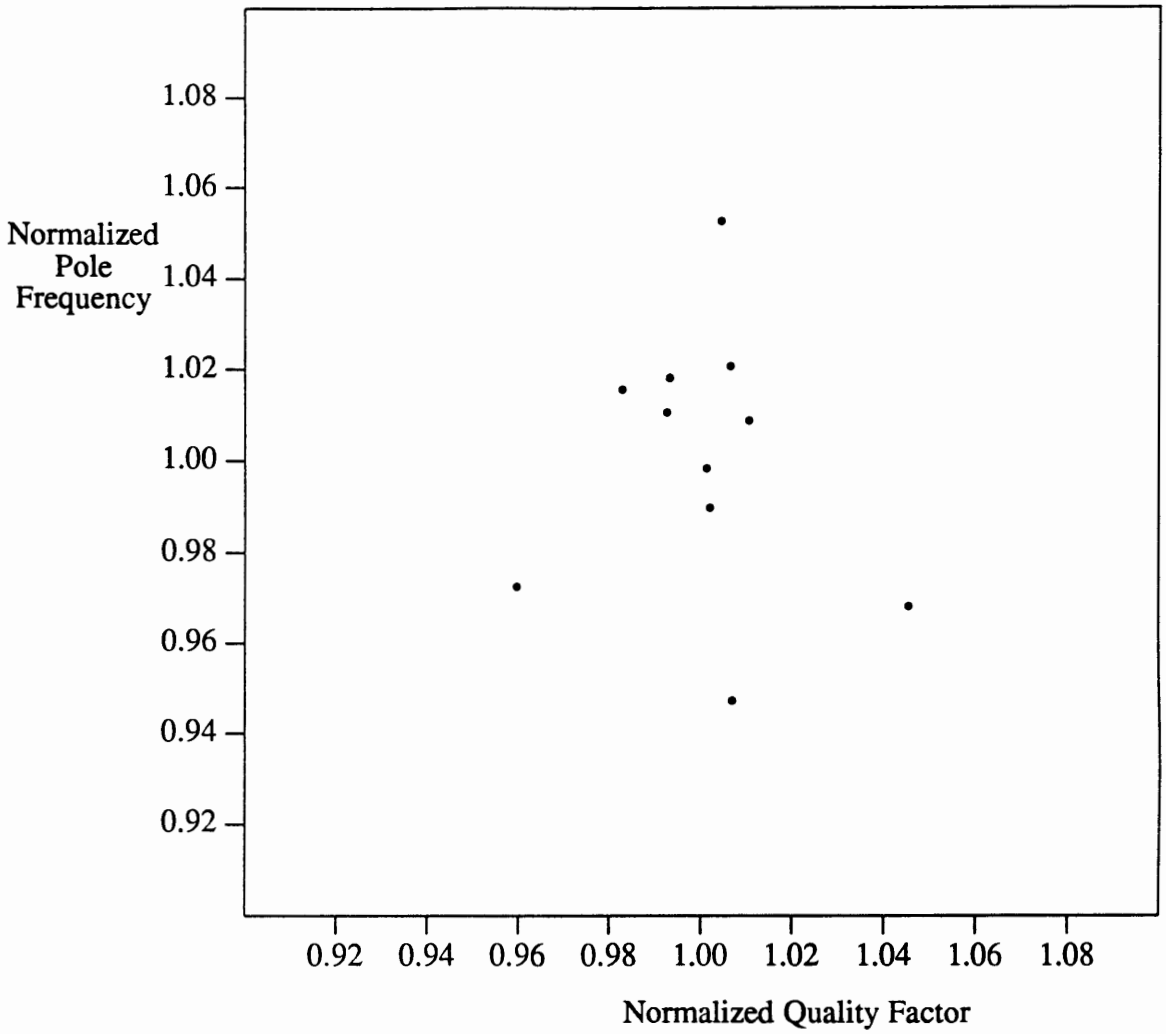


Figure 32. Residual errors of the quality factor Q_p and the pole frequency ω_p after tuning via NN (testing with 11 samples, generalization data, $Q_{p0} = 5.0$)(enlargement of Figure 31.).

V.3 ITERATIVE TUNING

In the previous section, we showed that the residual errors of the pole frequency and the quality factor, after the output of the NN has been used to tune the filter, remain almost in a 5% error-box around the nominal value. In other words, after the first iteration, the maximum error gets down to no more than about 5%. This is a big reduction. Remember, the initial error (Section IV.2) took on values up to 85%. However, the filter specifications may require even more accurate performance. To achieve a better performance, an iterative tuning scheme can be applied. The following sequence of experiments, for a nominal quality factor of 5.0, uses the first iteration and adds a second iteration to demonstrate that the control structure (Figure 10) is capable of performing tuning as an iterative process.

Once an iterative control structure is set up then a stopping mechanism must be developed. One approach is to include a threshold device in the control structure. This step is demonstrated in Figure 38. The task of the threshold device is to measure if the error-vector satisfies a given criterion. We assume that the filter specifications (provided by the customer) will stipulate that the error in the stopband(s) and the error in the passband(s) (Section II.1) have to stay below a specified threshold. Consequently, each component of the error-vector has to be checked with respect ot his threshold value. A requirement might be, for example that if the threshold condition for only one component of the error-vector is not satisfied, the iterative tuning process goes on. The relation between an error in the magnitude and changes in the pole frequency and the quality factor is schematically sketched in Figure 39.

Three components are involved in the tuning control strategy. One chip comprises the filter and the adjusting device. A second chip consists of the NN, a threshold device, a comparator, and a memory. Finally the digital spectrum analyzer measures the frequency samples and simultaneously supplies the filter with a spectrally rich signal.

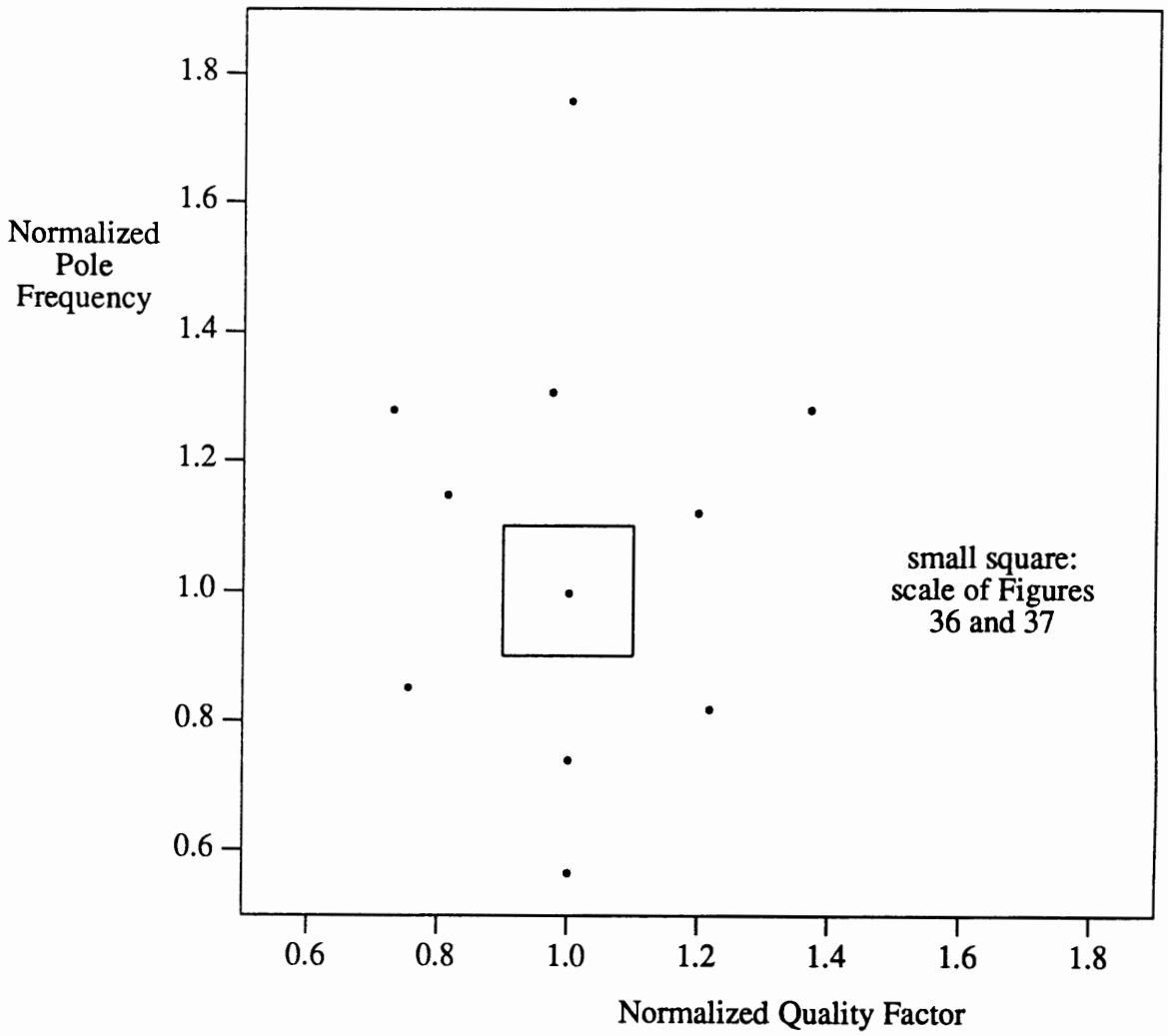


Figure 33. Initial errors of the quality factor Q_p and the pole frequency ω_p (11 samples, generalization data).

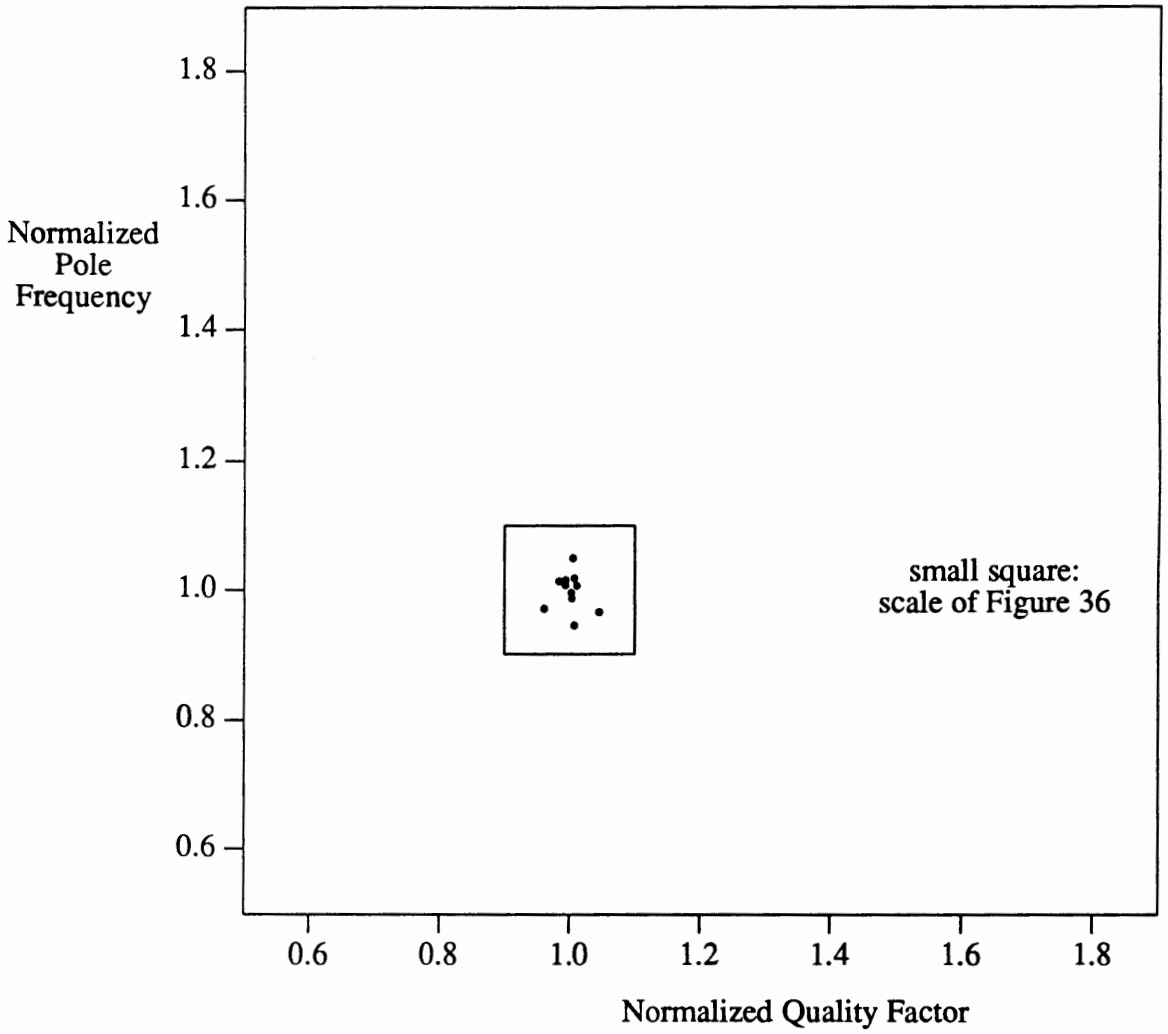


Figure 34. Residual errors of the quality factor Q_p and the pole frequency ω_p after tuning via NN (testing with 11 samples, generalization data,$Q_{p0} = 5.0$).

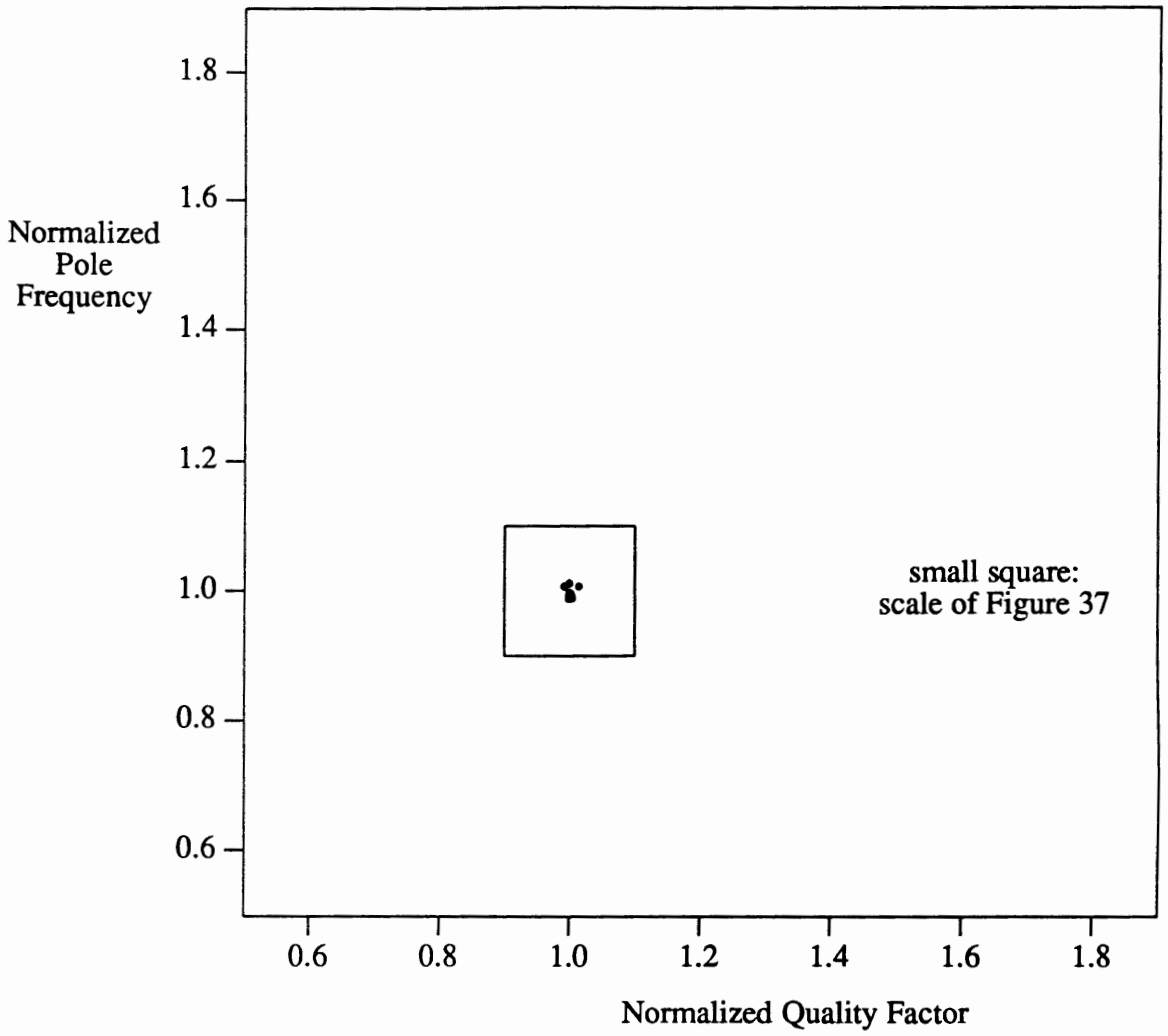


Figure 35. Residual errors of the quality factor Q_p and the pole frequency ω_p, after the output of the NN has been used to tune the filter the second time (11samples, generalization data,$Q_{p0} = 5.0$).

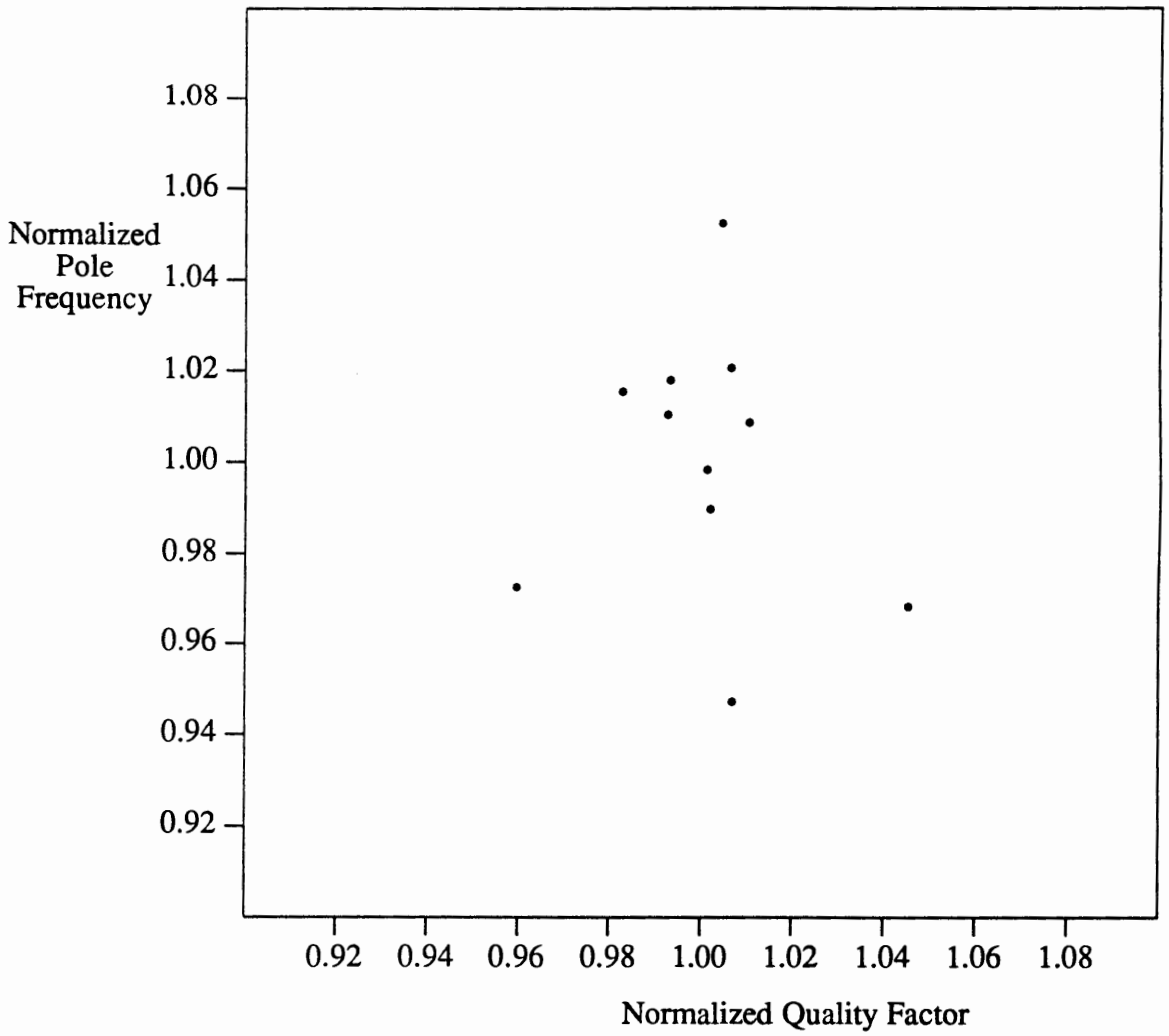


Figure 36. Residual errors of the quality factor Q_p and the pole frequency ω_p after tuning via NN (testing with 11 samples, generalization data, $Q_{p0} = 5.0$)(enlargement of Figure 34.).

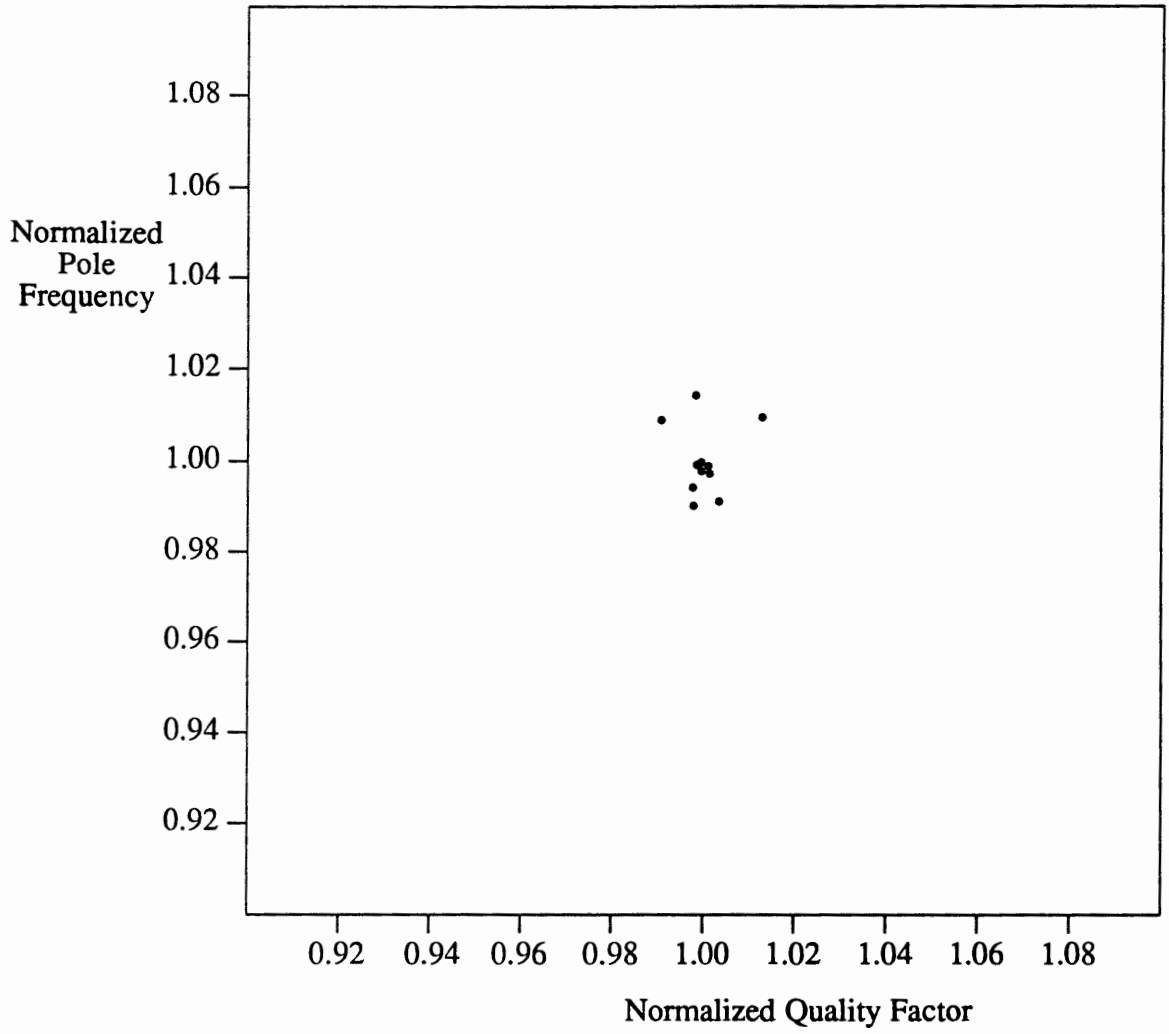


Figure 37. Residual errors of the quality factor Q_p and the pole frequency ω_p, after the output of the NN has been used to tune the filter the second time ($Q_{p0} = 5.0$, generalization data) (enlargement of Figure 35.).

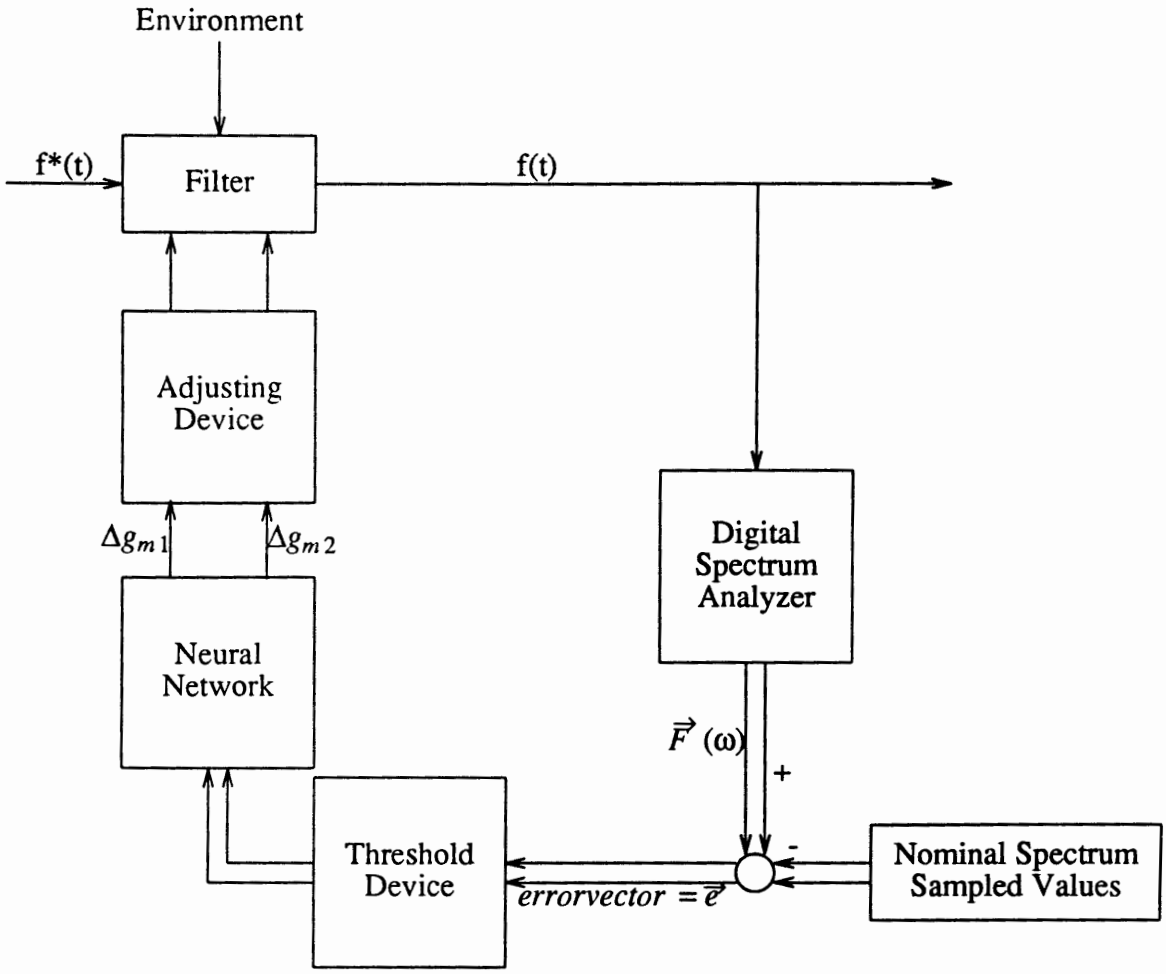


Figure 38. Control structure.

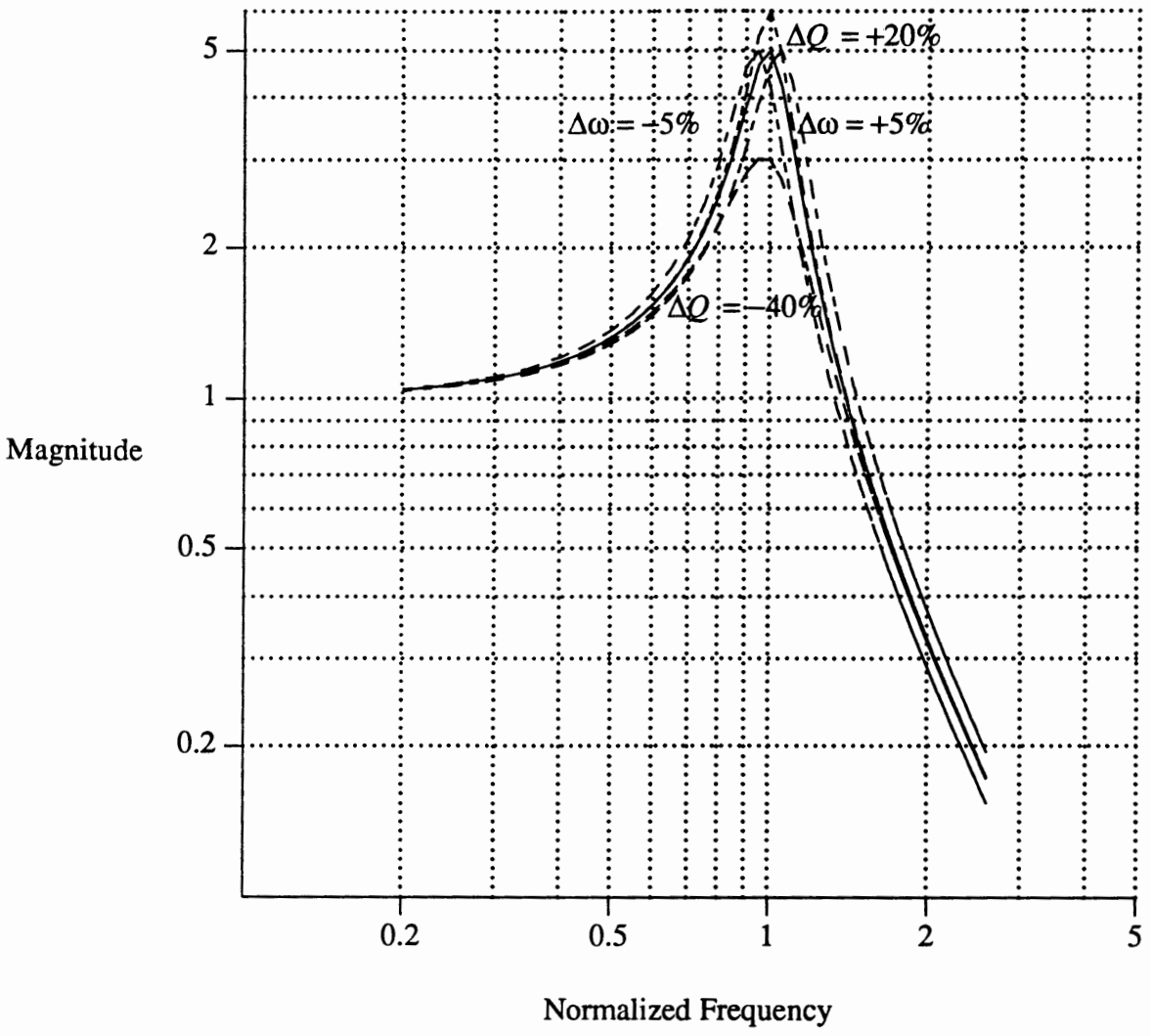


Figure39. Varying parameters and resulting changes in the magnitude.

V.4 TABULATED PERFORMANCE

TABLE IV

RESULTS OF TRAINING DATA (6253 RECORDS)

Q_{p0}	Q_{min}	Q_{max}	ω_{min} $10^7 Hz$	ω_{max} $10^7 Hz$	number of records in 5% range	number of records in 1% range	number of iterations (x1000)
5.0	4.707	5.292	0.936	1.066	6148	2113	200
3.33	3.137	3.503	0.938	1.066	6154	2254	300
2.5	2.354	2.618	0.938	1.065	6185	2297	200
1.67	1.593	1.744	0.935	1.062	6181	2306	200
1.25	1.195	1.306	0.939	1.063	6171	2332	200
1.0	0.949	1.046	0.942	1.062	6195	2316	200
0.83	0.789	0.871	0.940	1.072	6146	2313	250
0.707	0.662	0.743	0.944	1.072	6147	2233	250
0.625	0.582	0.656	0.938	1.072	6135	2195	300

TABLE V

RESULTS OF UNSEEN DATA (4896 RECORDS)

Q_{p0}	Q_{min}	Q_{max}	ω_{min} $10^7 Hz$	ω_{max} $10^7 Hz$	number of records in 5% range	number of records in 1% range	number of iterations (x1000)
5.0	4.798	5.227	0.948	1.053	4883	2022	200
3.33	3.200	3.475	0.947	1.052	4882	2131	300
2.5	2.398	2.605	0.948	1.052	4885	2171	200
1.67	1.605	1.742	0.950	1.054	4887	2179	200
1.25	1.200	1.304	0.950	1.051	4890	2194	200
1.0	0.959	1.050	0.950	1.048	4896	2203	200
0.83	0.794	0.870	0.942	1.050	4877	2166	250
0.707	0.675	0.741	0.941	1.047	4873	2094	250
0.625	0.600	0.653	0.941	1.050	4874	2128	300

CHAPTER VI

RESULTS FOR CIRCUIT A

These results are included to show that the simple Backpropagation network with only one hidden layer (Figure 16) is able to tune more than two filter components. Moreover a proposal is given to determine the number of components that can be tuned by this NN.

VI.1 INITIAL EXPERIMENT

The first experiments were done with Circuit A, with only the inductor L and the capacitor C varied. The nominal pole quality factor equalled the value 5.0. The range for changes was also smaller, from -20% to +20%. Tables VI states the average Euclidean distance in $L-C$-space (percentage), with respect to the value they were supposed to be. Table VII shows the the best outputs of the NN (Figure 16). Neither component is more than 1.5% off, after the output of the NN has been used to tune the filter. The NN accomplishes its task very accurately. As consequence of a smaller training set (81 records), the hidden layer consisted only of 9 PEs.

VI.2 NON-UNIQUE INVERSE MAPPING

In the beginning, the NN received 49 inputs, as this was the case in all experiments that have been run so far. It turned out that the NN was not able to determine the changes for three components. The maximum Euclidean distance took on values around 54%. The components varied between +30% and -30%, as given through the problem definition. Thus the NN did not figure out the input-ouput relationship. Going through some mathematics, the reason for this behavior became clear. Since a second-order

transfer function with unity gain is determined by only two parameters, different changes in the components cause the same error.

TABLE VI

LEARNING DYNAMICS
(PERCENTAGE ERROR IN L-C-SPACE)

iterations	2000	4000	6000	8000	10000	12000	20000
training	2.43	2.08	1.71	2.13	1.80	1.75	1.77
generalization	2.19	1.56	1.22	1.54	1.20	1.19	1.21

TABLE VII

PERFORMANCE
(PERCENTAGE ERROR IN L-C-SPACE)

	training	generalization
average	0.84	0.78
max	1.71	1.22
ΔC_{min}	-1.08	-0.40
ΔC_{max}	1.50	1.21
ΔL_{min}	-1.36	-1.20
ΔL_{max}	0.58	0.60

From a mathematical point of view the problem is characterized by the fact that we have fewer equations than unknowns. The determination of a unique solution requires as many equations as unknowns. In our case the missing, third equation is provided by a dc measurement of the resistor R (see Section IV.1). Indeed this additional information solved the problem of a non-unique mapping, such that the NN could encode the input-

output mapping. The results presented in the following sections will demonstrate the latter statement.

VI.3 RESULTS WITH 50 INPUTS

As consequence of the additional information the NN, shown in Figure 16, had to be modified. The number of input PEs increased to 50 and the NN comprises three output PEs. Everything else remained as before.

The Euclidean distance (with respect ot the desired value) and the amount single components are off is used as evaluation criterion of the NNs performance. All result data presented are percentage numbers. Figures 40 to 42 show the learning dynamic for three different quality factors. The maximum Euclidean distance is plotted as a function of the number of iterations the NN was trained. Again a solid line represents the results of the previously seen records. The plots demonstrate that the learning process is relatively fast. After 100,000 to 300,000 iterations the NN converges.

Table VIII summarizes the performance of the NN, which was trained to tune Circuit A. The maximum Euclidean distance is approximately 5%. Single components are no more than 4% off. Thus the NN shows a good performance, even when three components are to be tuned.

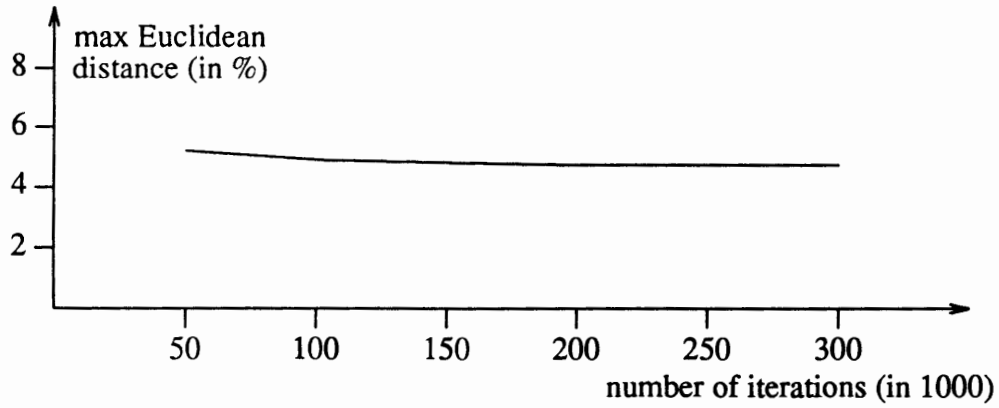


Figure 40. Network dynamics for $Q_{p0} = 5.0$.

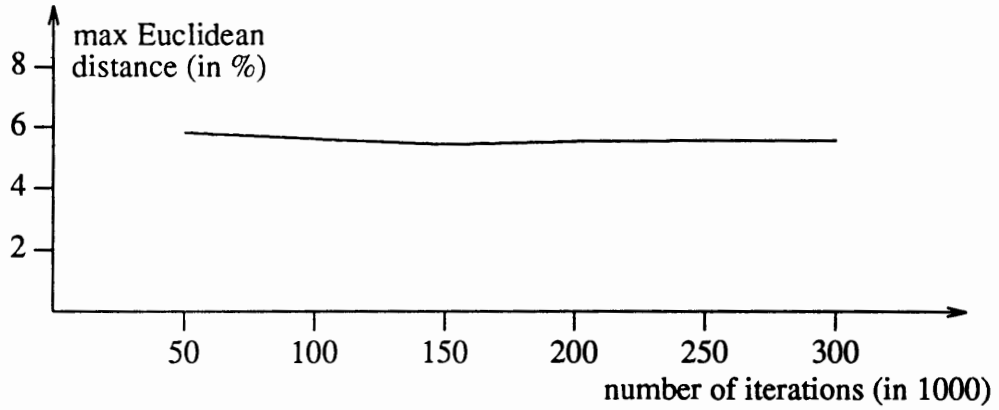


Figure 41. Network dynamics for $Q_{p0} = 1.0$.

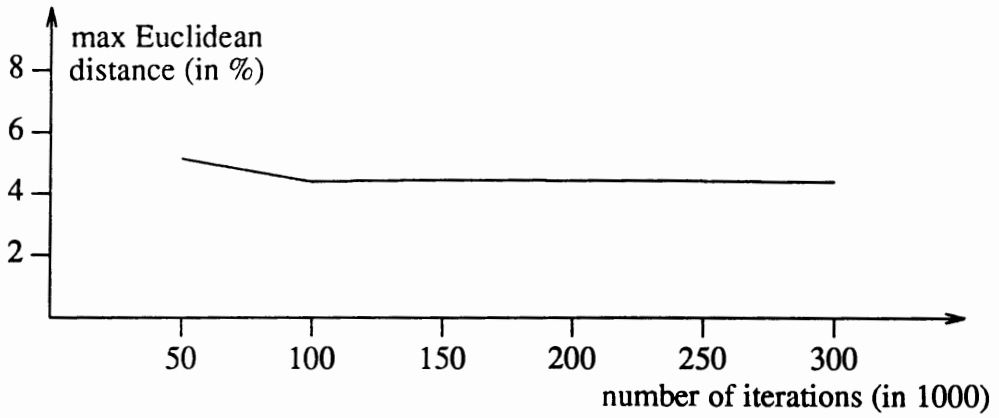


Figure 42. Network dynamics for $Q_{p0} = 0.625$.

TABLE VIII

RESIDUAL ERRORS AFTER TUNING (UNSEEN DATA)

Q_{p0}	5.0	1.0	0.625
average Euclidean distance (%)	1.14	1.21	0.85
max Euclidean distance (%)	3.57	3.84	3.07
$\Delta Rmin$ (%)	-2.04	-2.76	-1.25
$\Delta Rmax$ (%)	1.62	1.39	1.20
$\Delta Lmin$ (%)	-2.09	-3.00	-2.87
$\Delta Lmax$ (%)	1.84	2.63	1.57
$\Delta Cmin$ (%)	-3.57	-3.11	-2.64
$\Delta Cmax$ (%)	2.12	1.64	1.31

CHAPTER VII

CONCLUSIONS

The objective of this research was to demonstrate whether the Neural Network methodology could meaningfully assist in the task of tuning an active, continuous-time, analog filter, implemented via the transconductance-C-method. The results are good.

The approach developed herein measures the frequency response of the manufactured filter at 49 selected frequencies, calculates the differences from the designed (nominal) values at those frequencies, and inputs these 49 error values into the Neural Network (NN). The NN has two outputs for the filters implemented via the transconductance-C-method. These two outputs specify the amount of error in the two transconductances Δg_{m1} and Δg_{m2}.

The tuning control strategy would be to use these values of Δg_{m1} and Δg_{m2} by an adjusting device to modify values of the appropriate bias currents and/or voltages in the manufactured filter.

The NNs used in the experiments achieved a reduction of manufacturing errors of up to 85% for the pole frequency down to less than approximately 5%. It was demonstrated that the method can be iterated to further reduce the error. One extra iteration reduced the worst errors to less than 1.5%.

Experimental results provide a demonstration that the NNs used are capable of tuning filters to compensate manufacturing errors. Yet, parasitic effects might exist and cause errors in the magnitude of the transfer function. If the presence of the parasitic effects would lead to the fact that the values of the actual quality factor and the actual pole frequency are more off than the worst case training record (see Figure 26.), the

respective filter would have to be thrown away. However, one can expect that, even when parasitic effects are taken under consideration, most of the manufactured (actual) filters realize a quality factor and a pole frequency within the train-set. In these cases the NN will accomplish a good job. If it turns out that too many manufactured filters (e.g., 10%) realize a quality factor and a pole frequency outside the train-set, then the train-set should be enlarged, such that it comprises a more realistic representation of the actual tolerances of the quality factor and the pole frequency.

This method is applicable to a factory setting with off-line measurement and tuning. An important next step will be to determine whether the approach could be modified to function on-line, so as to implement a tuning capability that would compensate for parameter variations due to aging and/or environmental changes.

Even within the off-line context described herein, there remain a number of system parameters to explore further. For example, it may not be required to use 49 sample values of the frequency response. A series of experiments could be carried out to determine some optimum value of measurements needed.

This thesis investigated second-order filters, in particular low-pass filters. Realizing that Circuit B (Figure 6.) implements both a low-pass filter and a band-pass filter makes evident that tuning of the low-pass filter simultaneously corrects the pole frequency and the quality factor of the band-pass filter.

Further questions relate to higher-order filters. If implemented via the cascade of second-order modules (see equation 2.2), can we use the NN to tune each module separately? If implemented via other methods, can we use the NN methodology in a similar fashion for higher-order filters? A significant issue becomes that of developing training data to encompass variations of a large number of parameters -- a combinatorial issue.

This research provides an encouraging first step.

REFERENCES

[1] R. Schaumann, "The Design of Continuous-Time Fully Integrated Filters: A Tutorial," Integrated Continuous-Time Filters: Principles, Design and Applications, IEEE Circuits and Systems Society, A selected Reprint, 1993.

[2] R. Schaumann, M.S. Ghausi, and K.R. Laker, "Design of Analog Filters: Passive, Active RC, and Switched Capacitor," Englewood Cliffs, N.J., Prentice-Hall, 1990.

[3] R. Schaumann and M.A. Tan, "The Problem of On-Chip Automatic Tuning in Continuous-Time Integrated Filters," Proceedings IEEE International Symposium on Circuits and Systems, 1989.

[4] F. Krummenacker and N. Joehl, "A 4-MHz CMOS Continuous-Time Filter with On-Chip Automatic Tuning," IEEE Journal of Soild-State Circuits, June 1988.

[5] K.W. Moulding, J.R. Quartly, P.J. Rankin, R.S. Thompson, and G.A. Wilson, "Gyrator Video Filter IC with Automatic Tuning," IEEE Journal of Solid-State Circuits, December 1980.

[6] P.M. Van Peteghem and R. Song, "Tuning Strategies in High-Frequency Integrated Continuous-Time Filters," IEEE Transactions on Circuits and Systems, January 1989.

[7] Karen A. Kozma, David A. Johns, and Adel S. Sedra, "Automatic Tuning of Continuous-Time Filters Using an Adaptive Filter Technique, IEEE Transactions on Cirsuits and Systems, Vol. CAS-38, pp. 1241-1248, November 1991.

[8] G.C. Temes and J.W. LaPatra, "Introduction to Circuit Synthesis and Design," McGrawhill, 1977.

[9] James A. Freeeman and David M. Skapura, "Neural Networks," Addison-Wesley Publishing Company, 1992.

[10] Maureen Caudill, "Avoiding the Great Backpropagation Trap," Neural Network Special Report, 1992.

[11] Arthur D. Hall, "Metasystems Methodology," Pergamon Press, Oxford, New-York, 1989.

[12] John Hertz, Anders Krogh, and Richard G. Palmer, "Introduction to the Theory of Neural Computation," Addison-Wesley Publishing Company, 1992.

[13] Donald O. Hebb, "The Organization of Behavior," Wiley, New-York, 1949.

[14] Bart Kosko, "Neural Networks and Fuzzy Systems," Prentice Hall, 1992.

[15] NeuralWorks Professional II/Plus, "Neural Computing," NeuralWare Incorporation, 1991.

[16] NeuralWorks Professional II/Plus, "Reference Guide," NeuralWare Incorporation, 1991.

[17] Russel D. Reed and Randall L. Geiger, " A Multiple-Input OTA Circuit for Neural Networks," IEEE Transactions on Circuits and Systems, Vol. 36, May 1989.

[18] Paul W. Hollis and John J. Paulos, "An Analog BiCMOS Hopfield Neuron," Analog Integrated Circuits and Signal Processing, An International Journal, Kluwer Academic Publishers, November 1992.

[19] Seokjim Kim, Yong-Chul Skin, Naida C.R. Bogineni and Ramglingam Sridhur, "A Programmable Analog CMOS Synapse for Neural Networks," Analog Integrated Circuits and Signal Processing, An International Journal, Kluwer Academic Publishers, November 1992.

[20] Jeanette Lawrence, "Data Preparation for a Neural Network," Neural Network Special Report, 1992.

[21] Alan Lapedes and Robert Faber, "How Neural Nets Work," Neural Information Processing Systems, 1987.

[22] R. Schaumann, "The Design of Continuous-Time Fully Integrated Filters: A Review, IEE Proceedings, Vol. 136, Electronic Circuits and Systems, pp. 184-190, August 1989.

[23] Tom Kwan and Kenneth Martin, "An Adaptive Analog Continuous-Time CMOS Biquadratic Filter," IEEE Journal of Solid-State Circuits, Vol. SC-26, pp.859-867, 1991.

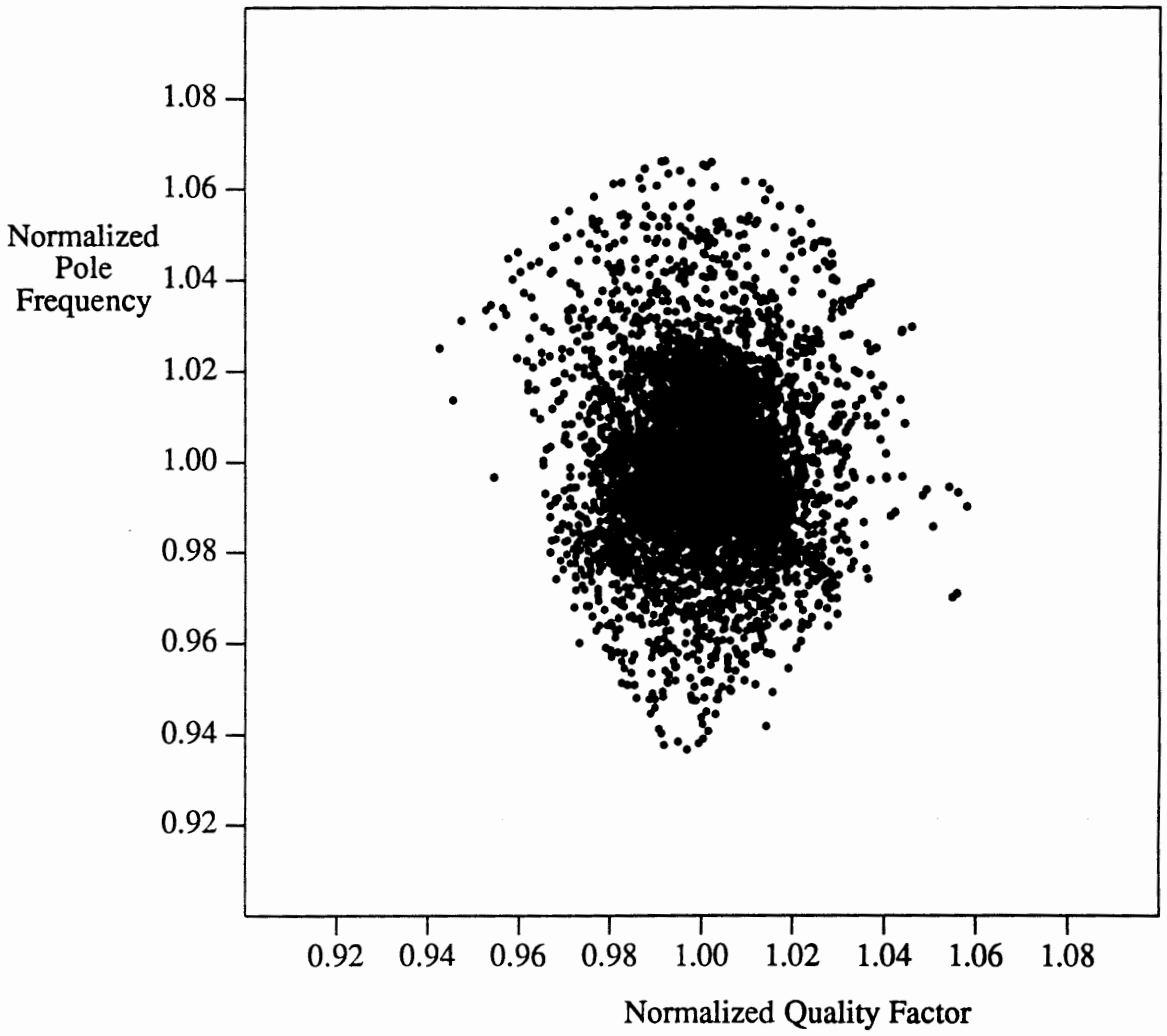
[24] Edgar Sanchez-Sinencio, Randall L. Geiger, and H. Nevarez-Lozano, "Generation of Continuous-Time Two Integrator Loop Filter Structures," IEEE Transactions on Circuits and Systems, Vol. 35, August 1988.

[25] Jacek M. Zurada, "Analog Implementation of Neural Networks," IEEE Transactions on Circuits and Systems, September 1992.

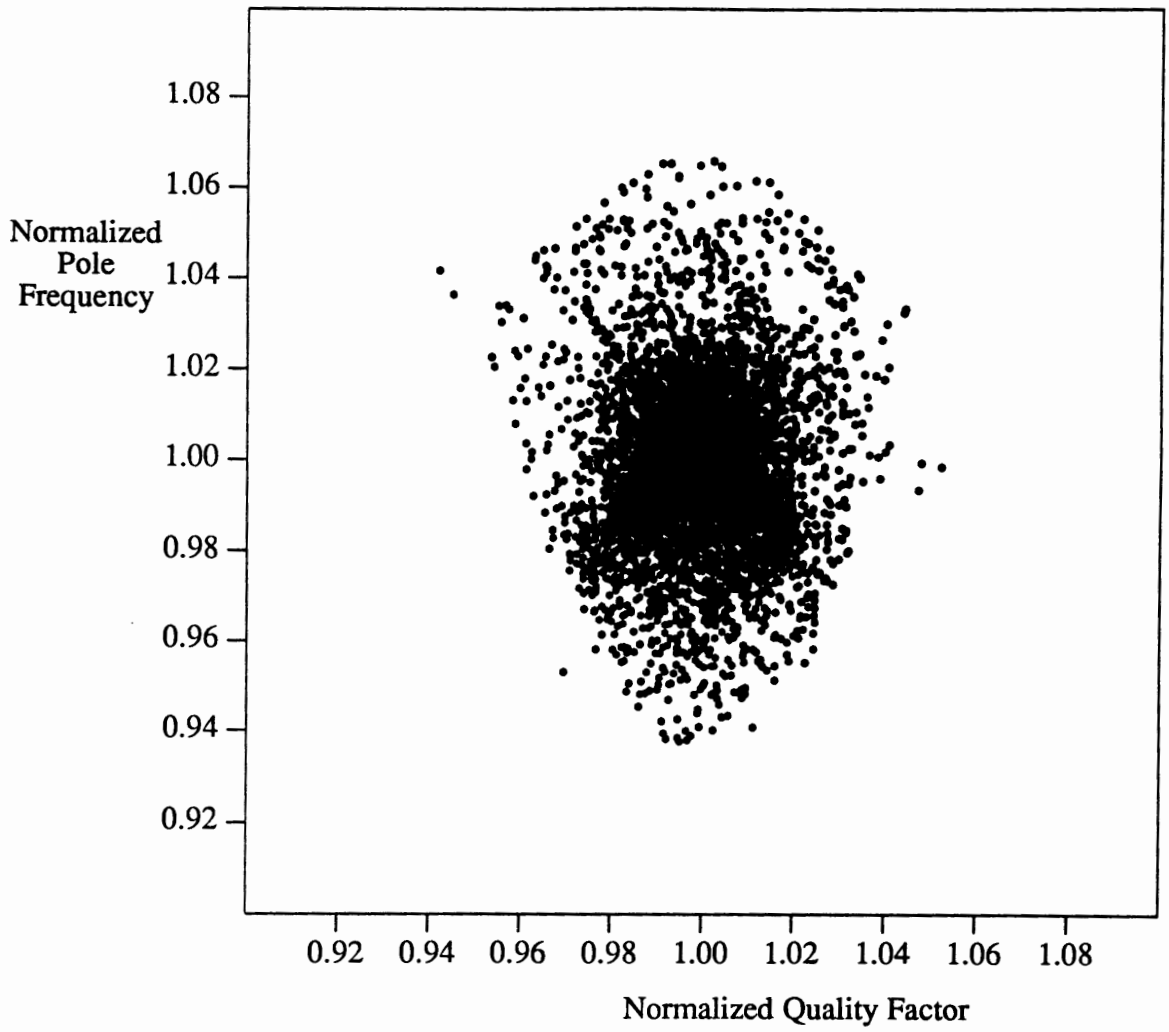
APPENDIX A

RESULTS OF CIRCUIT B'S NEURAL NETWORK

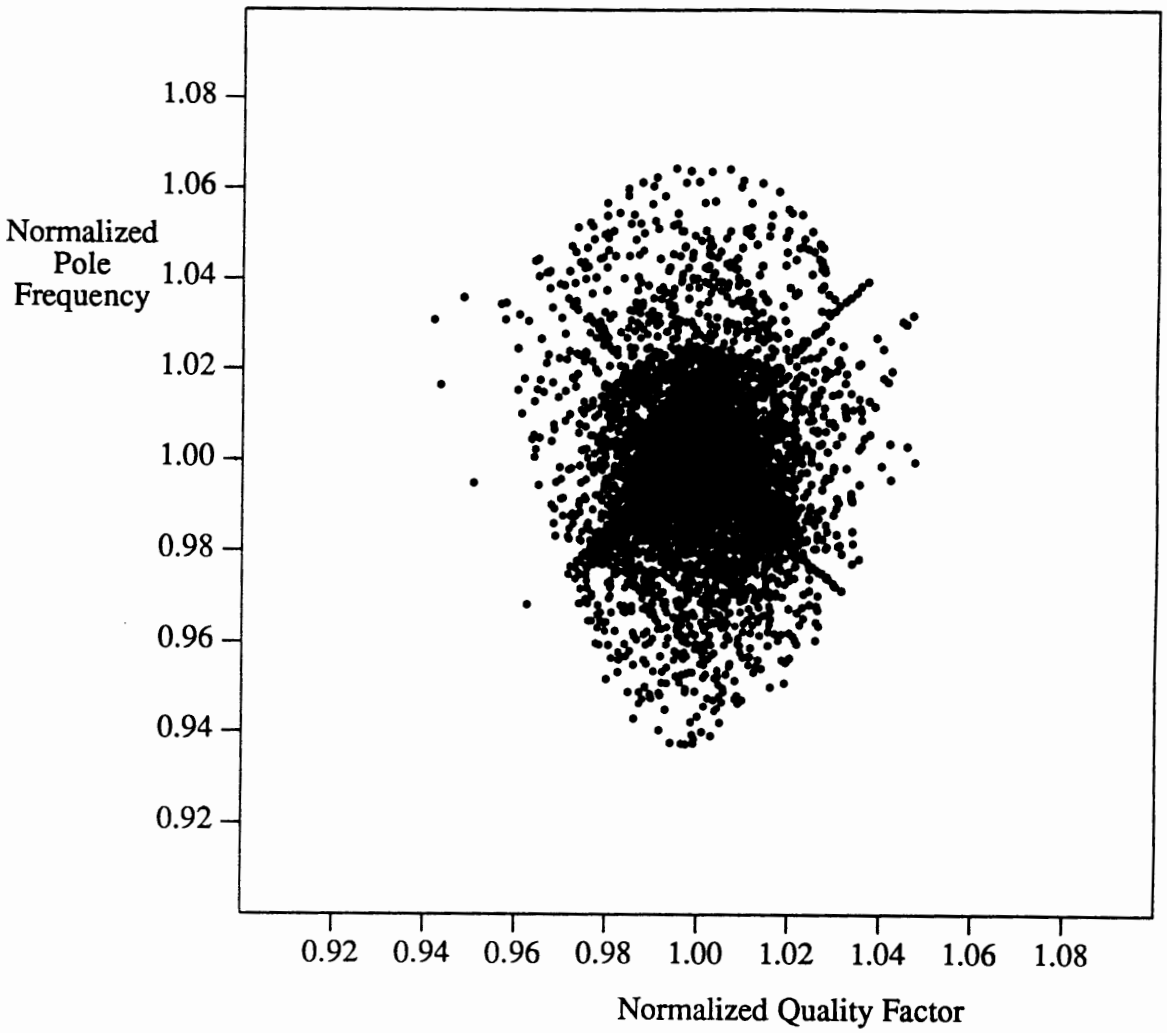
A.1. ENLARGEMENTS OF RESIDUAL ERRORS OF TRAINING DATA



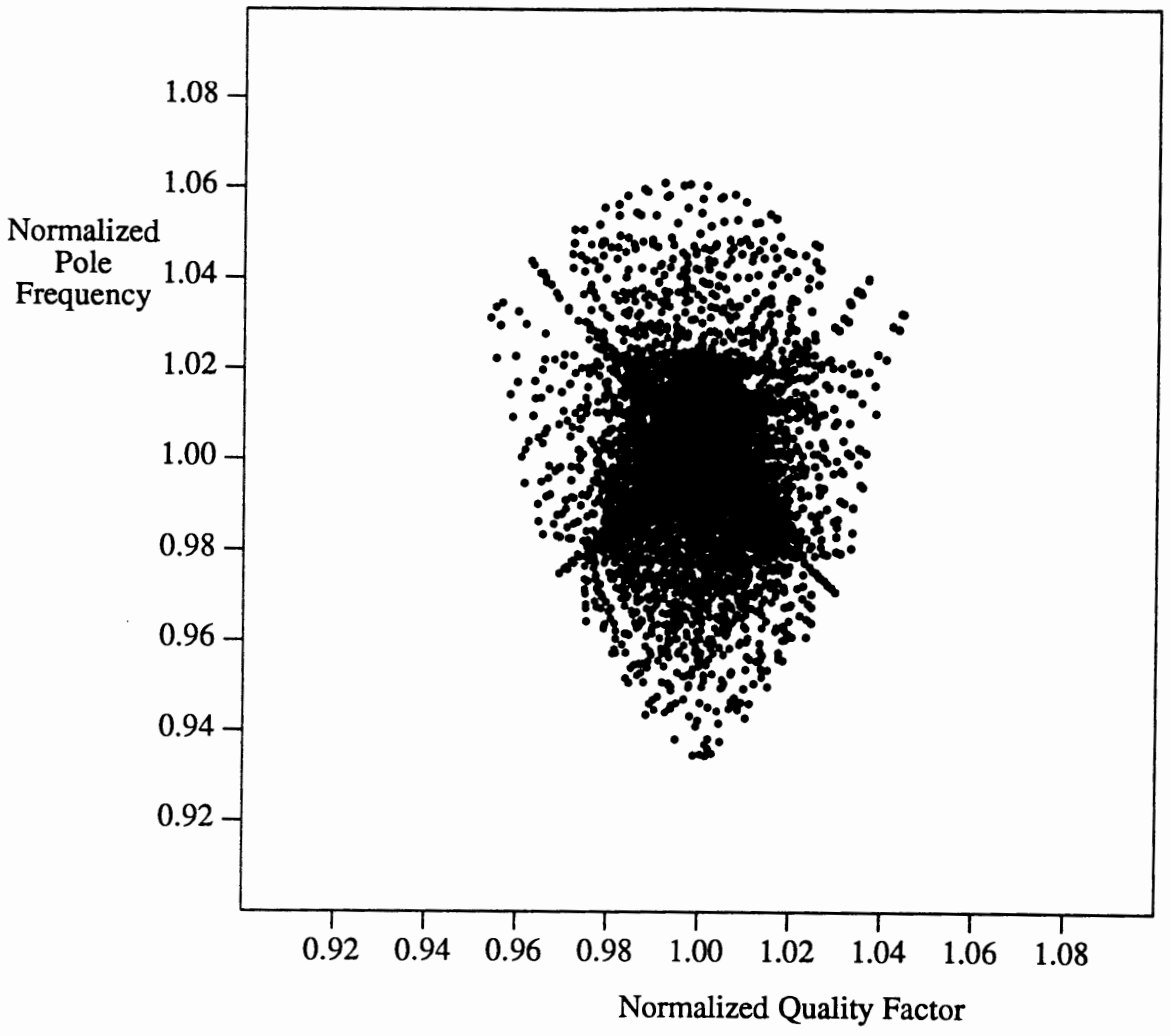
A1. Residual errors of Q_p and ω_p after tuning via NN
($Q_{p0} = 5.0$, training data)



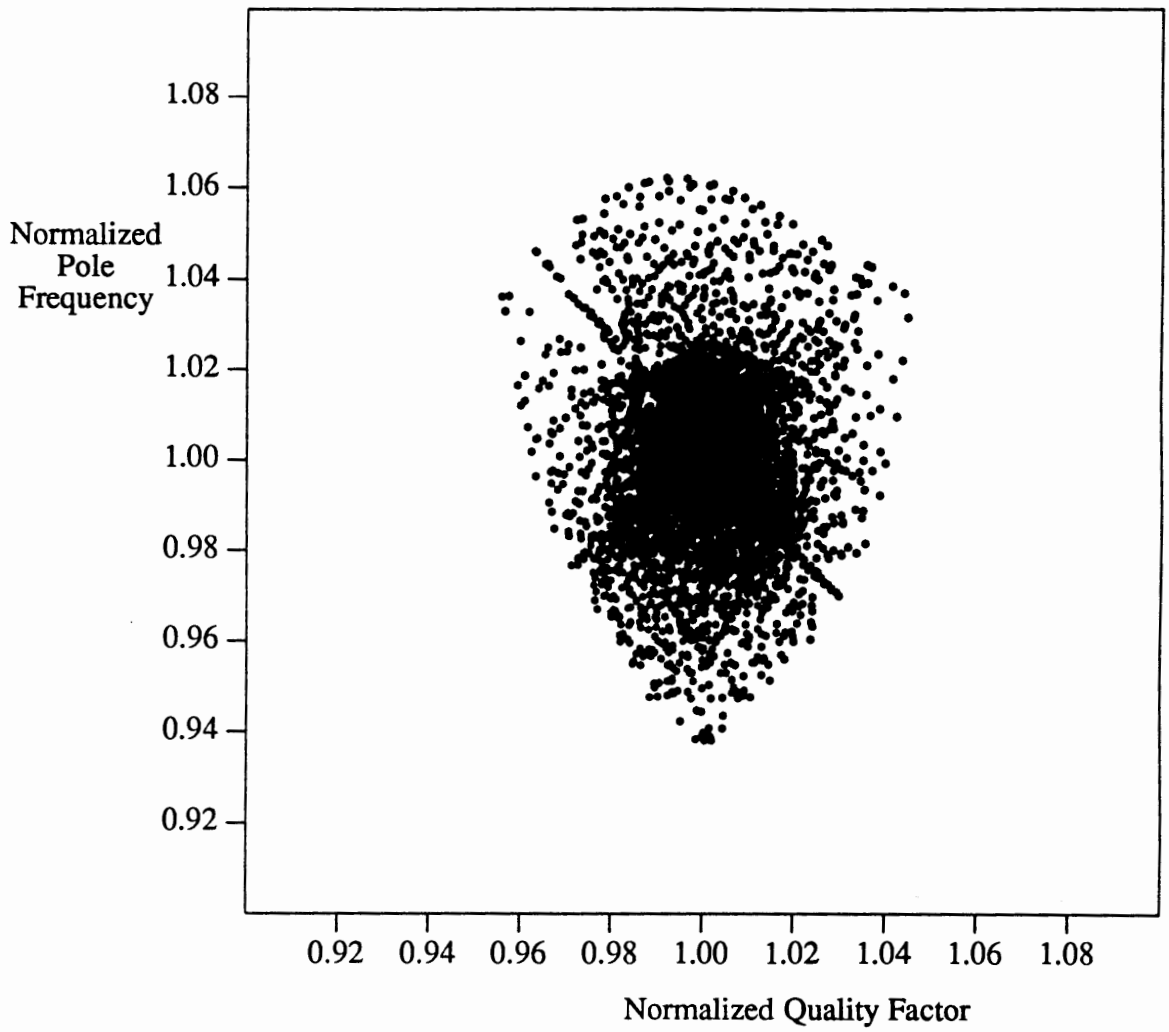
A2. Residual errors of Q_p and ω_p after tuning via NN
($Q_{p0} = 3.33$, training data)



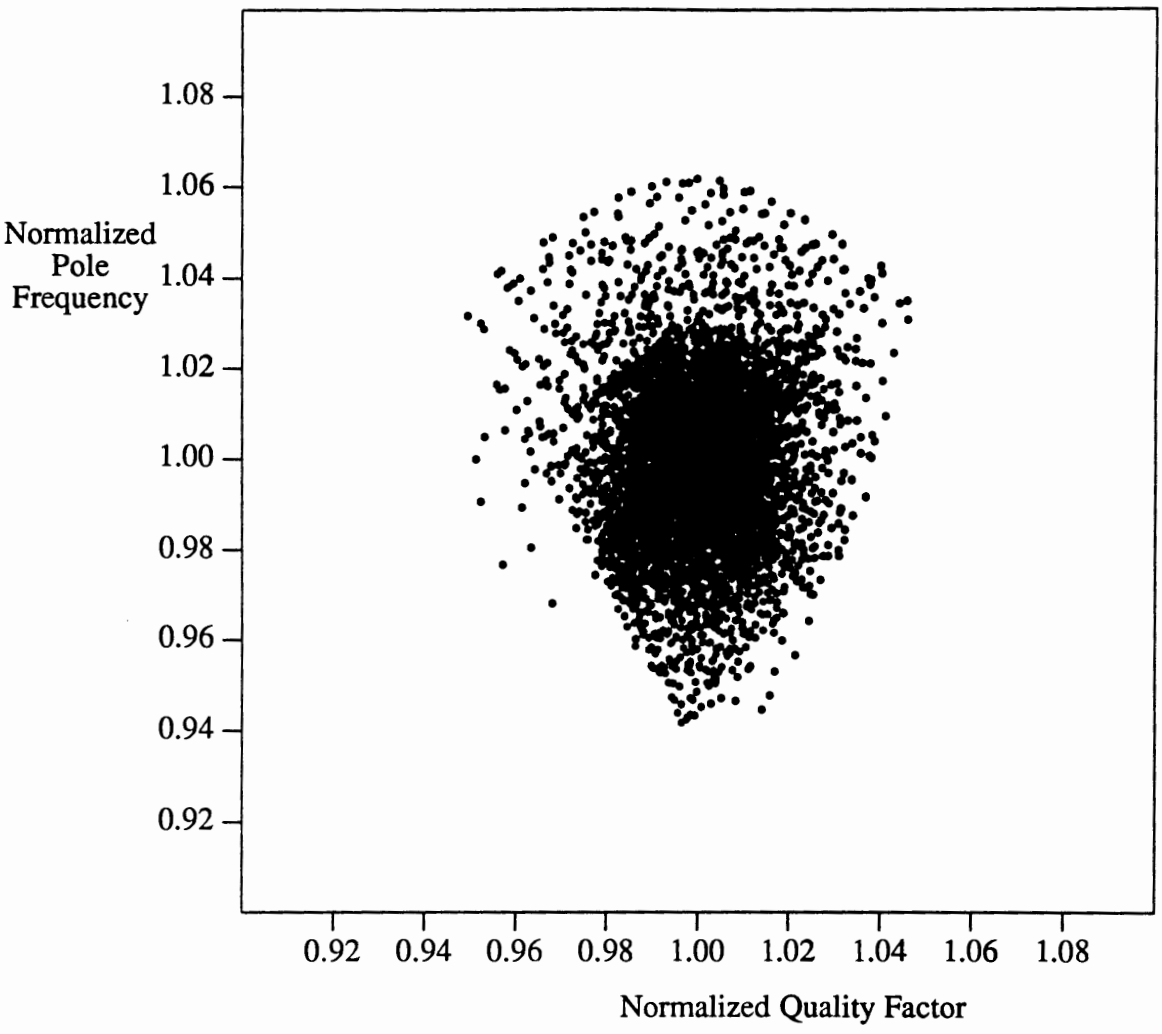
A3. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 2.5$, training data)



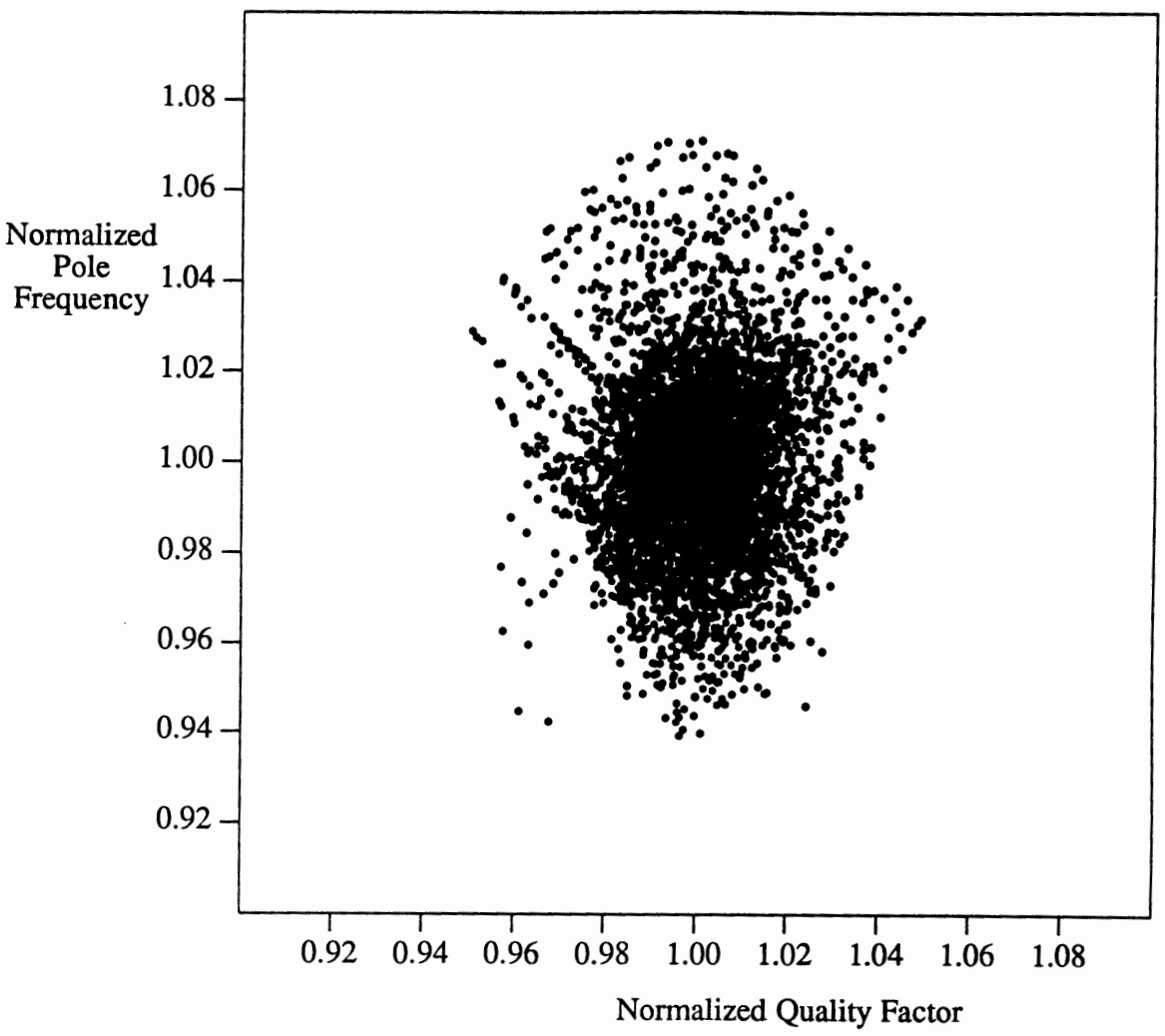
A4. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 1.67$, training data)



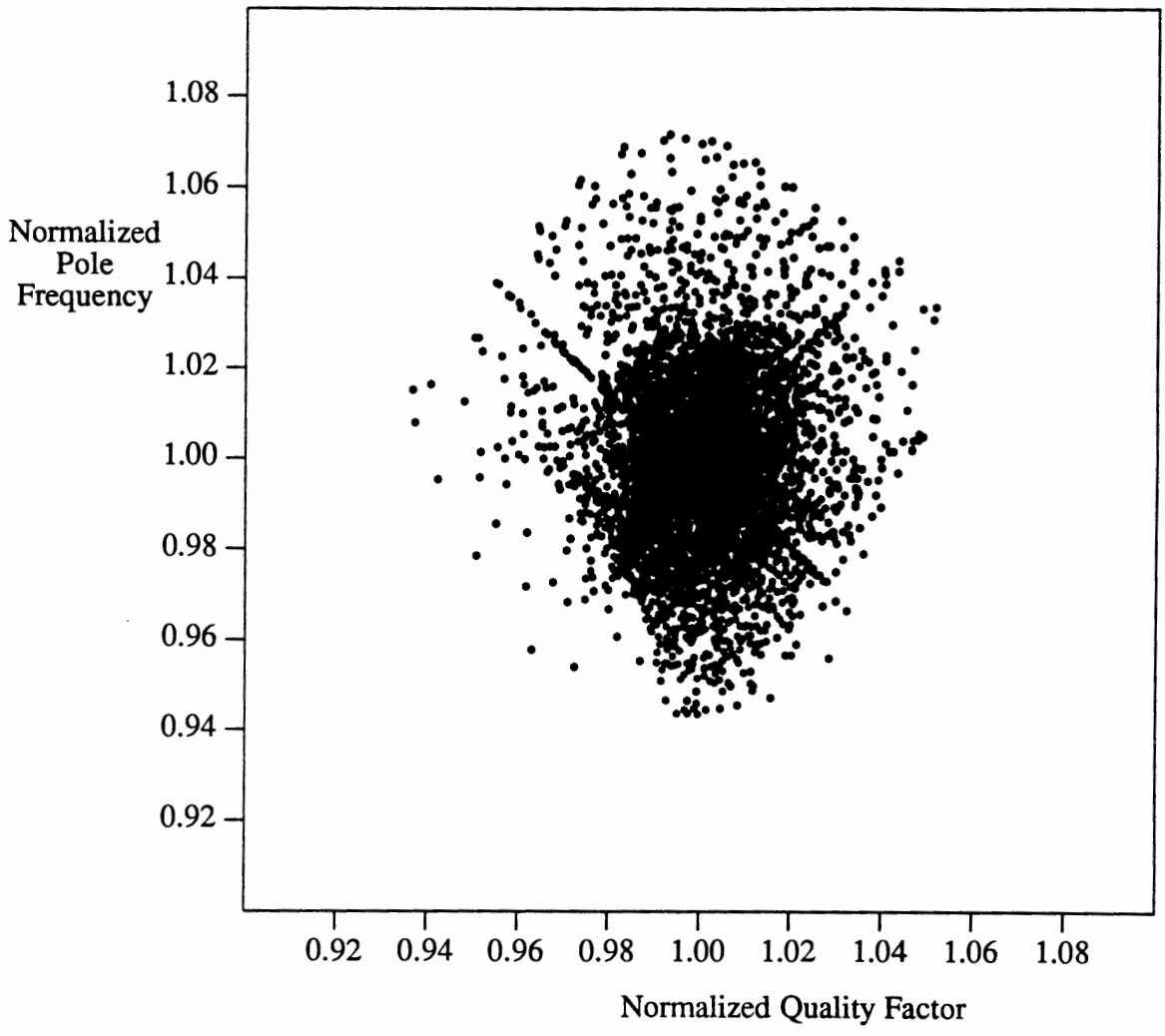
A5. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 1.25$, training data)



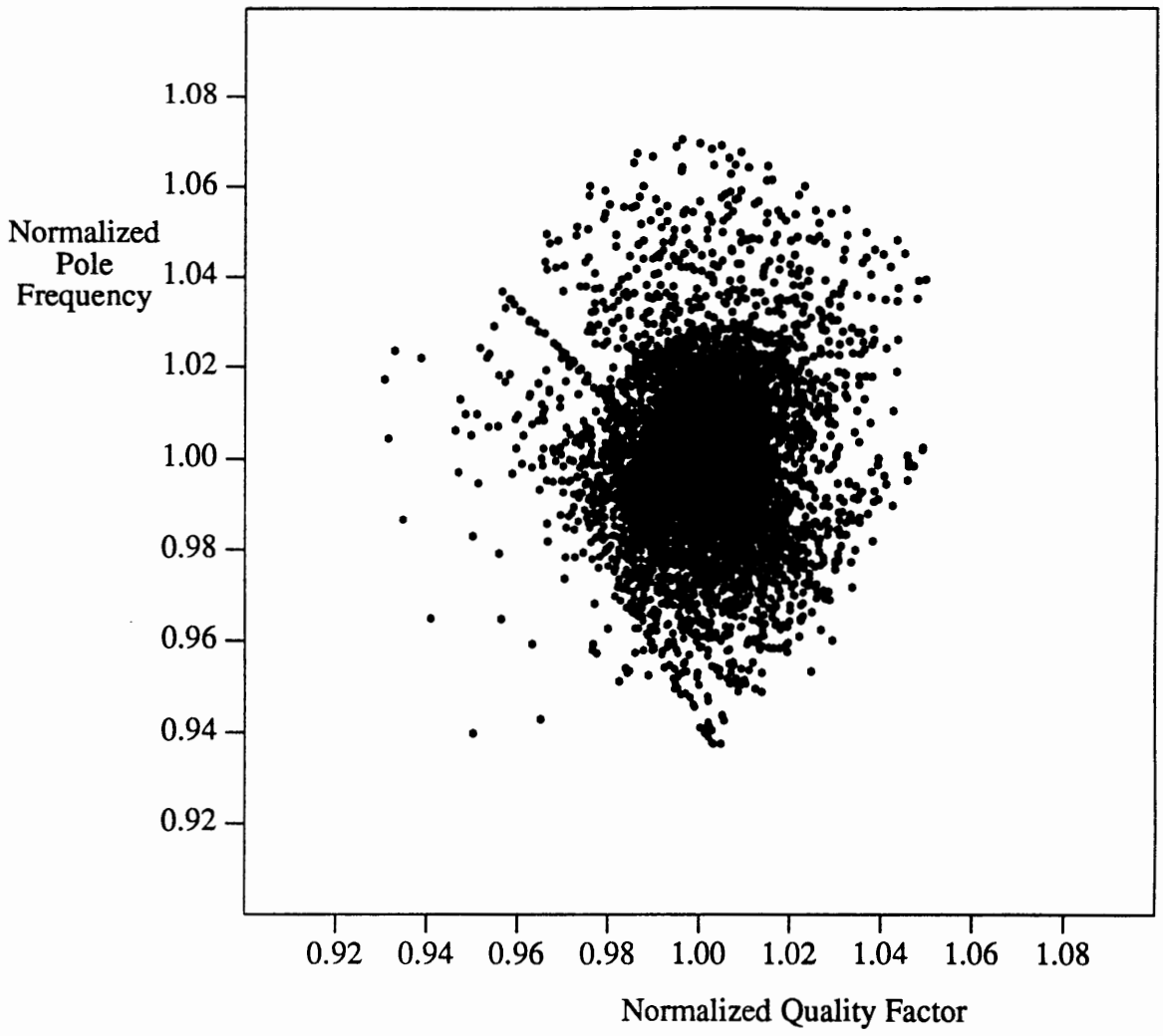
A6. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 1.0$, training data)



A7. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 0.83$, training data)

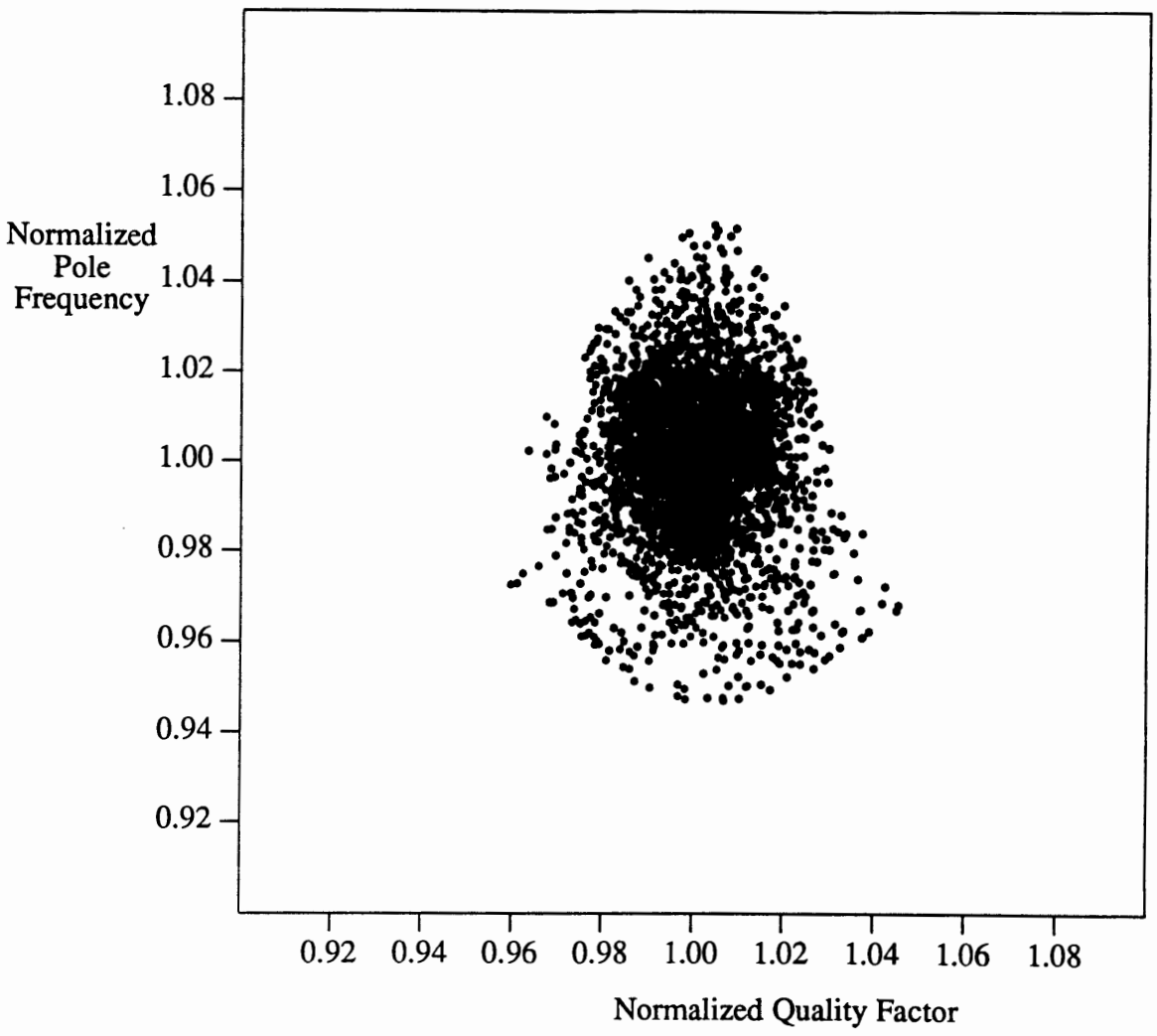


A8. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 0.707$, training data)

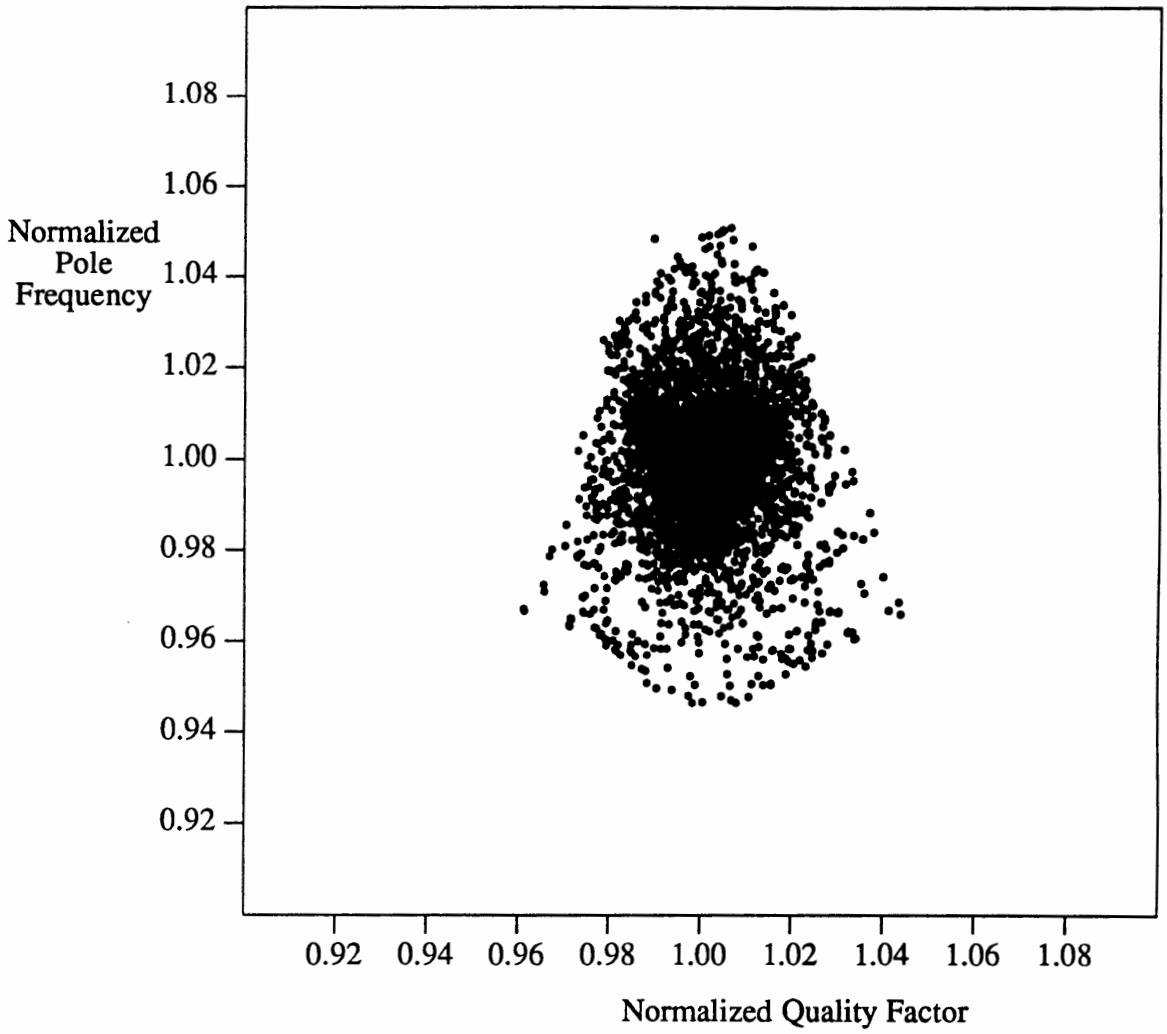


A9. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 0.625$, training data)

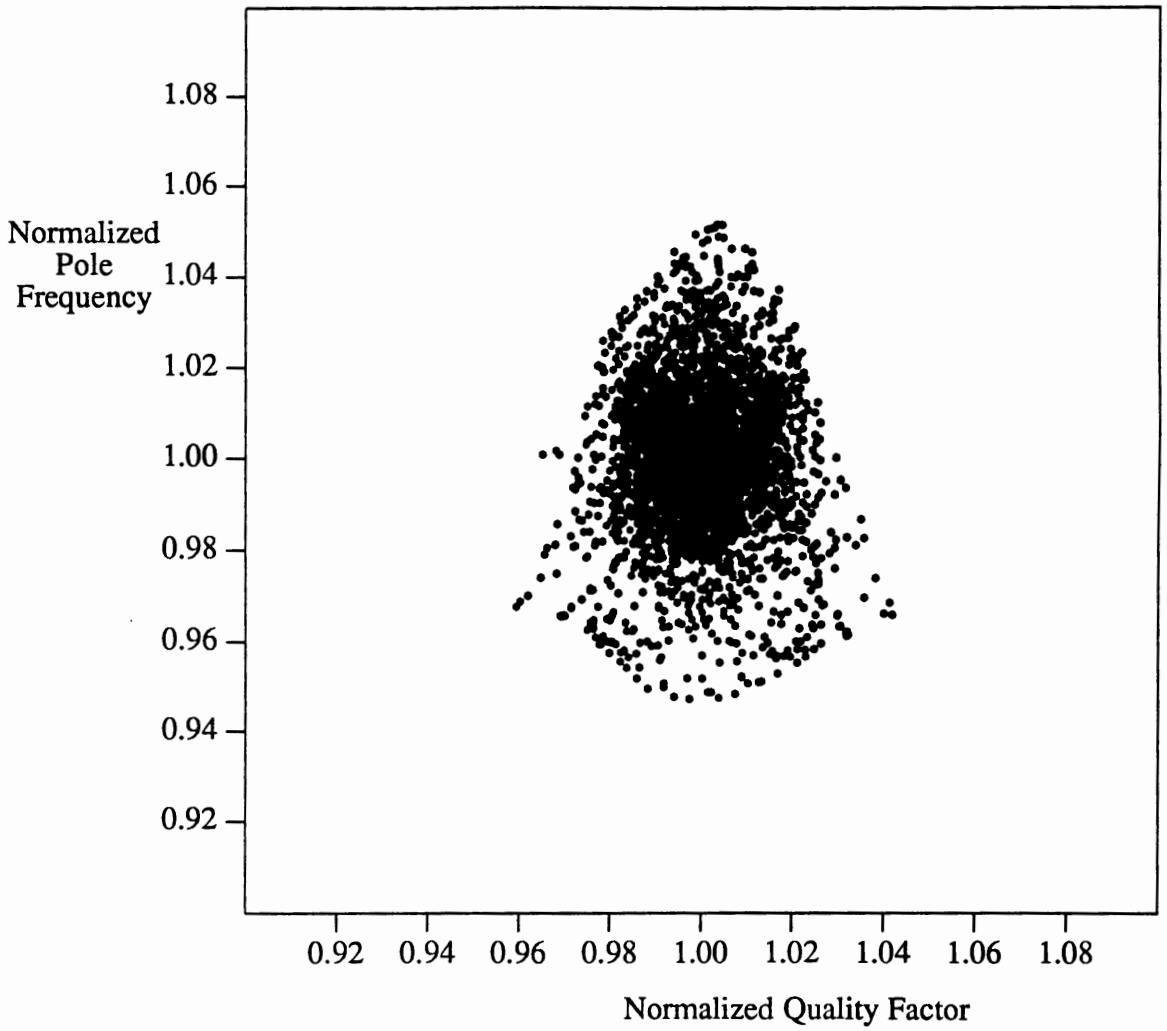
A.2. ENLARGEMENTS OF RESIDUAL ERRORS OF GENERALIZATION DATA



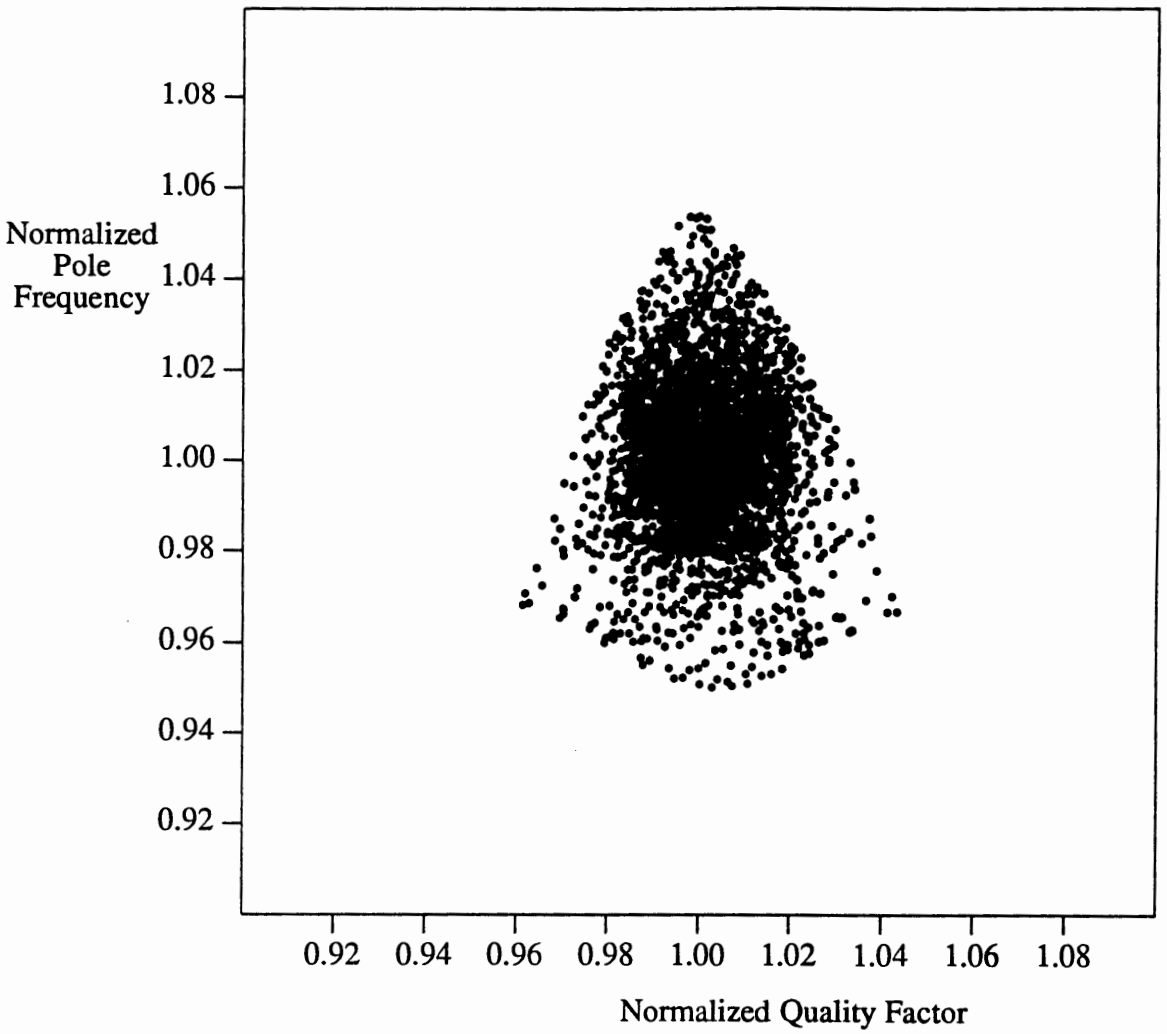
A10. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 5.0$, generalization data)



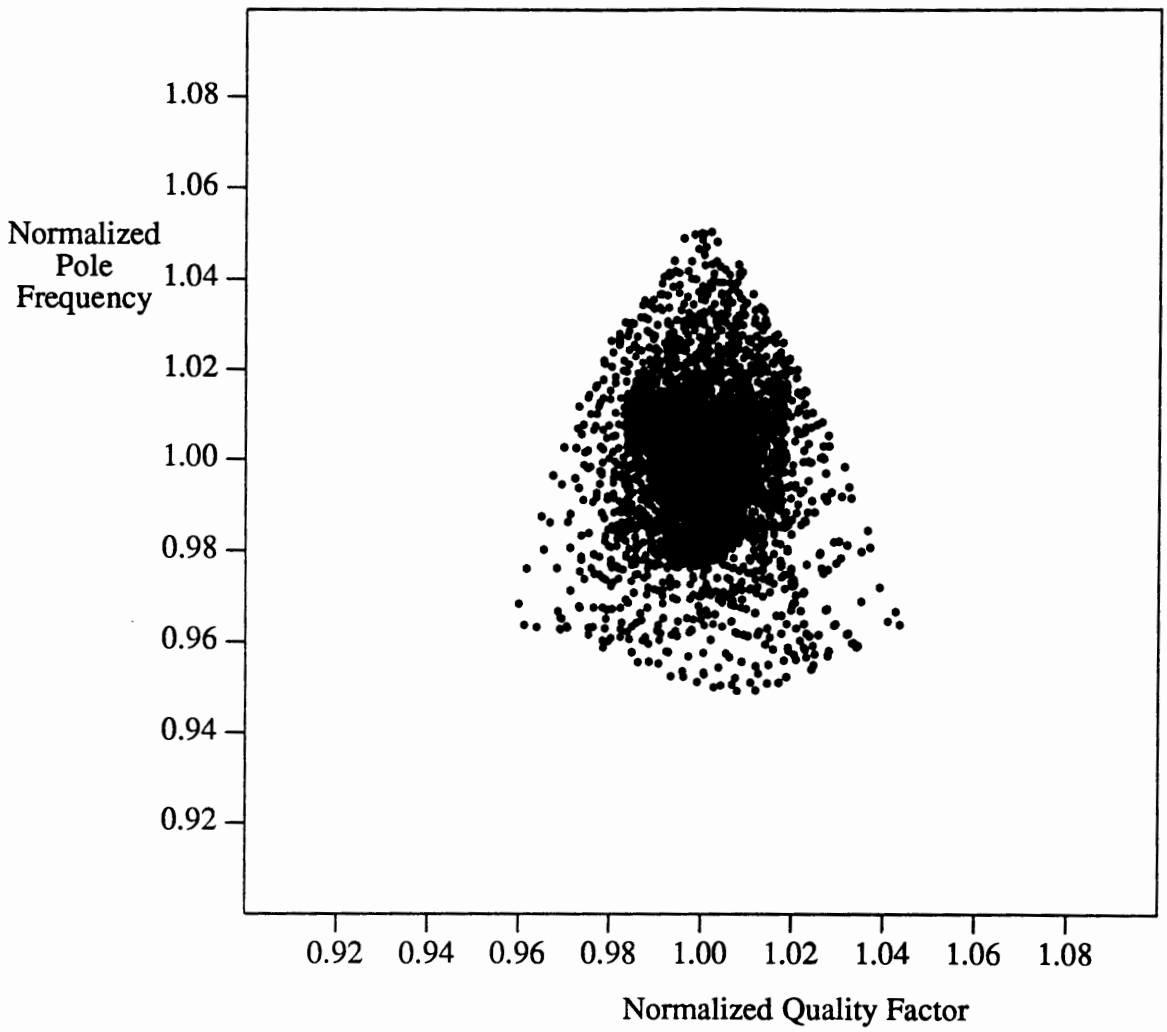
A11. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 3.33$, generalization data)



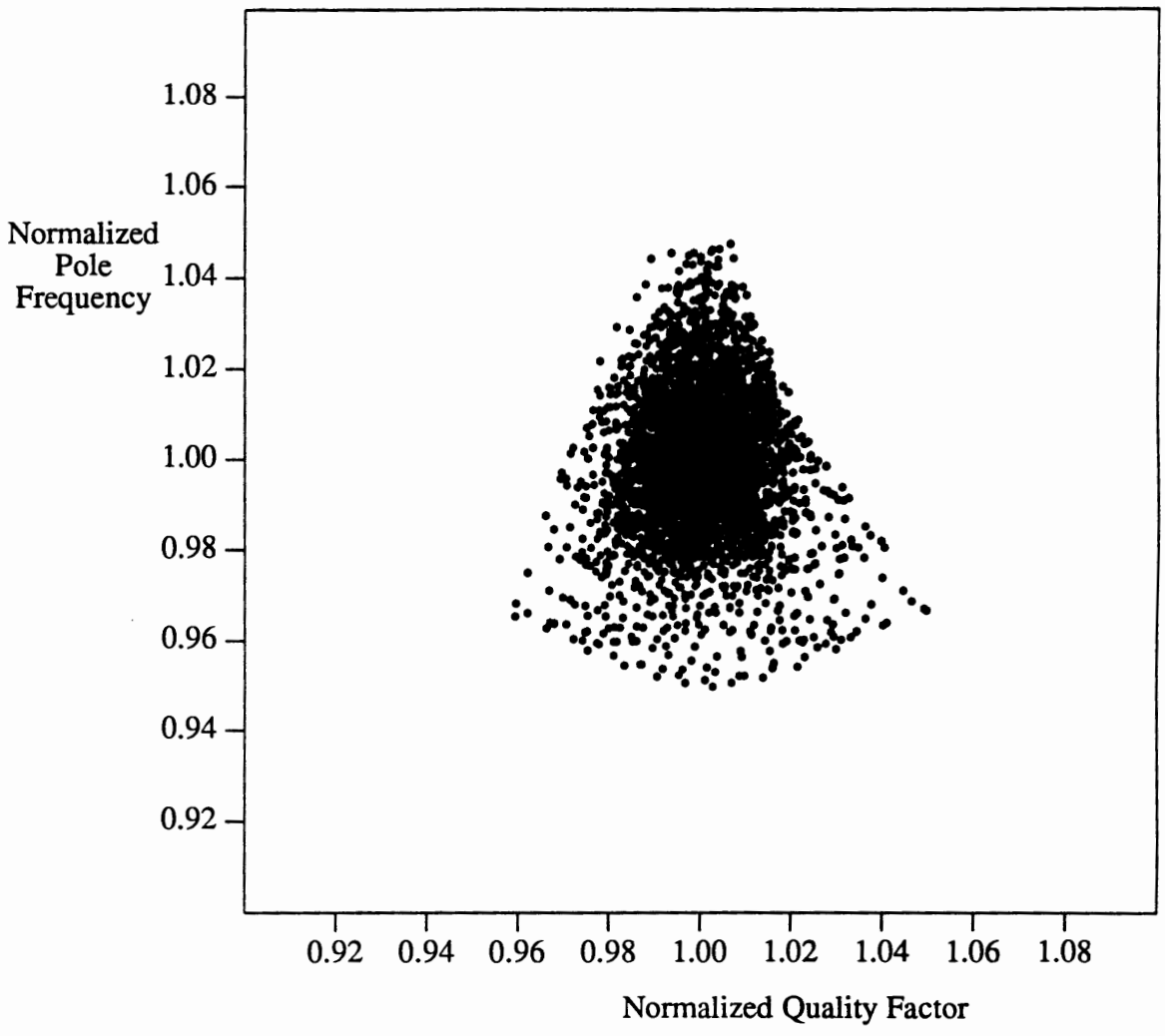
A12. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 2.5$, generalization data)



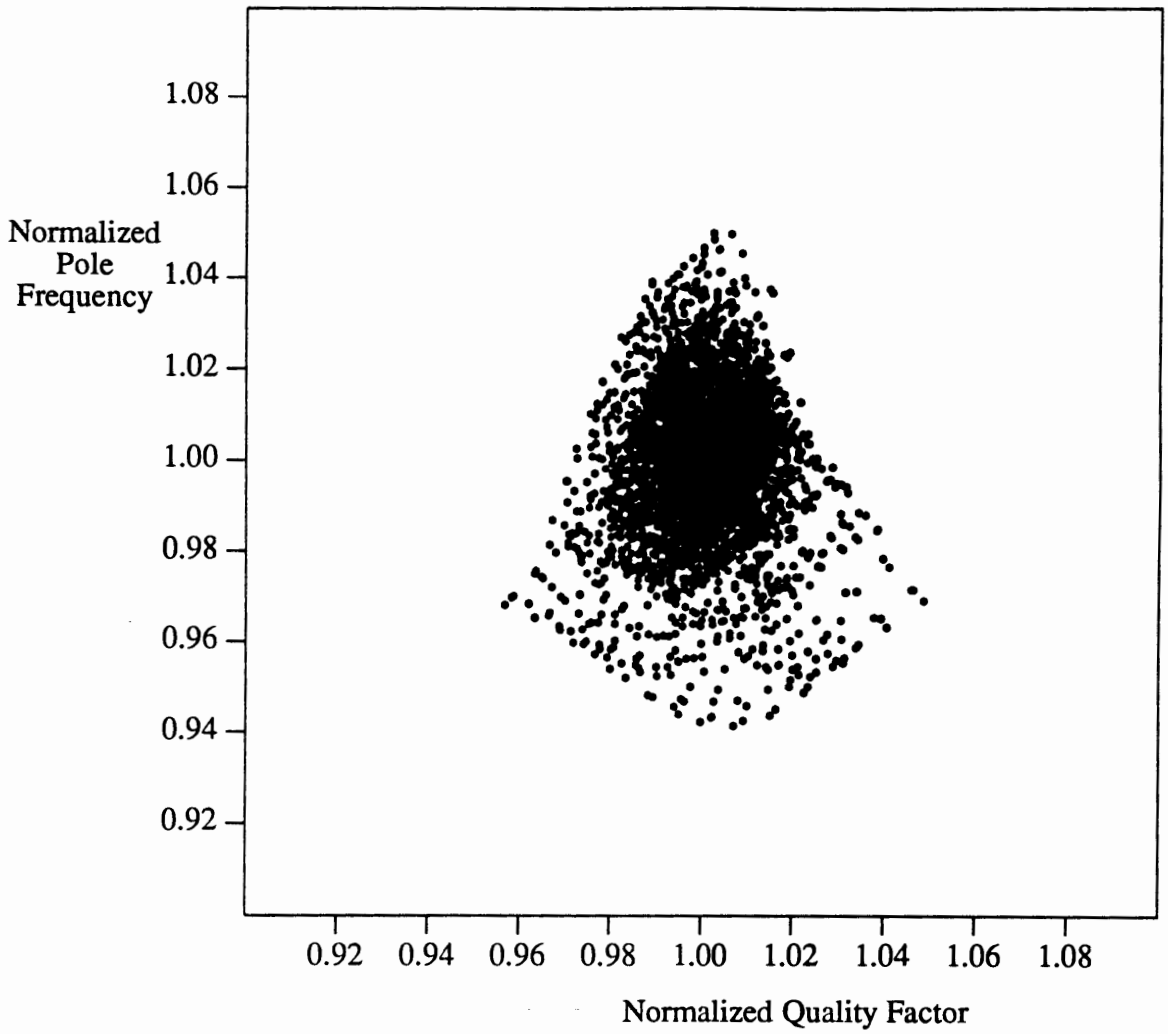
A13. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 1.67$, generalization data)



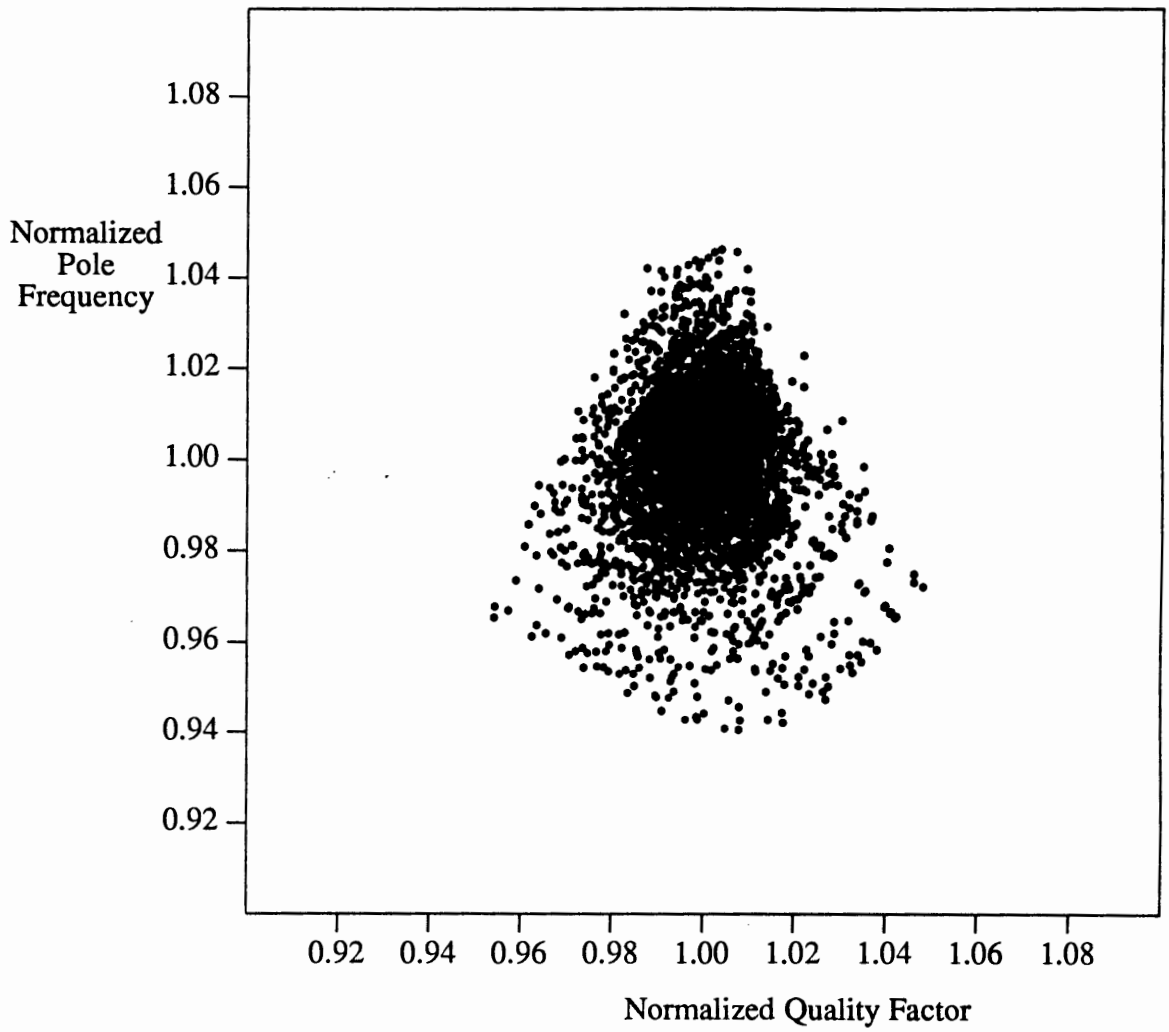
A14. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 1.25$, generalization data)



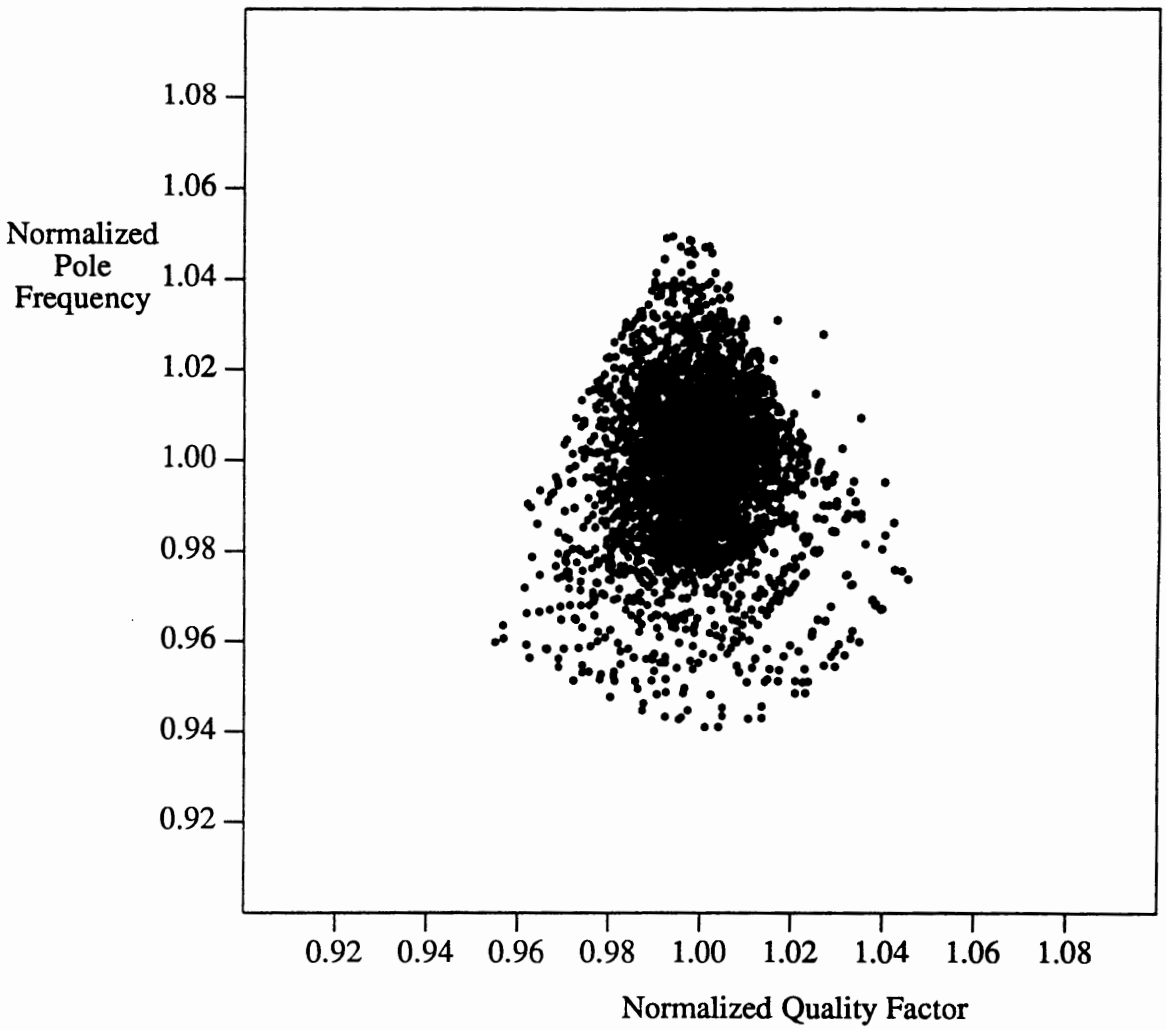
A15. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 1.0$, generalization data)



A16. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 0.83$, generalization data)

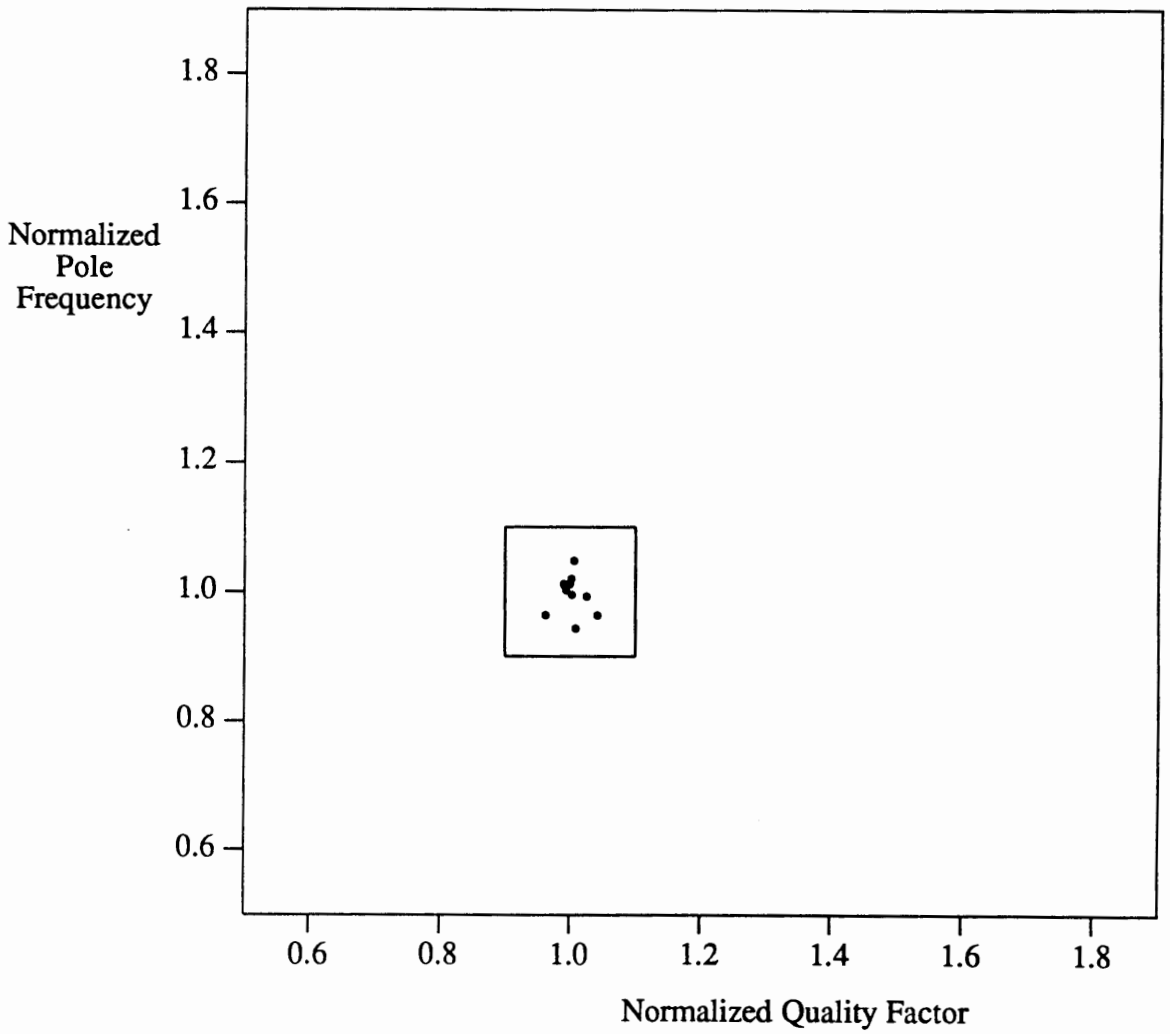


A17. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 0.707$, generalization data)

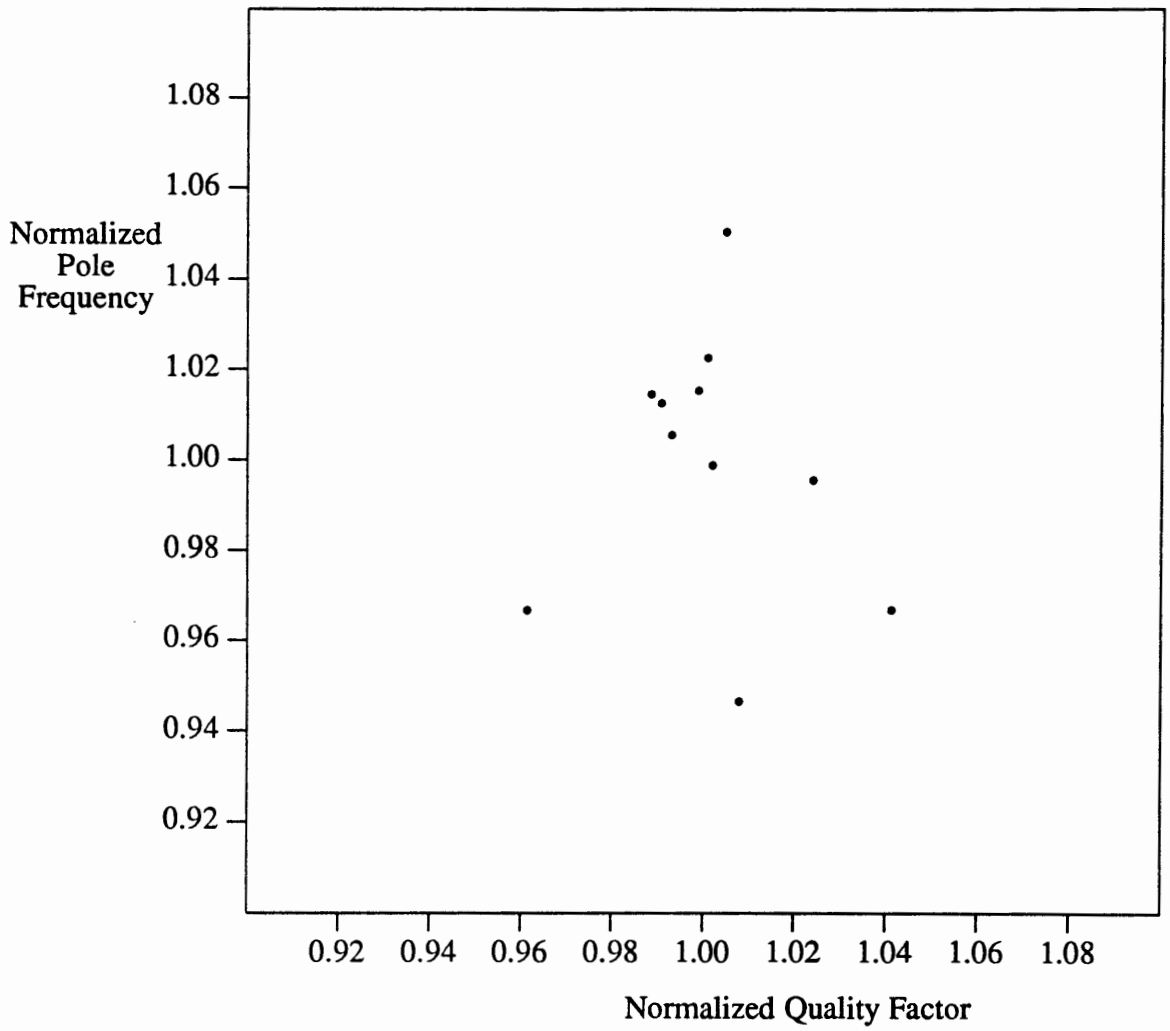


A18. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 0.625$, generalization data)

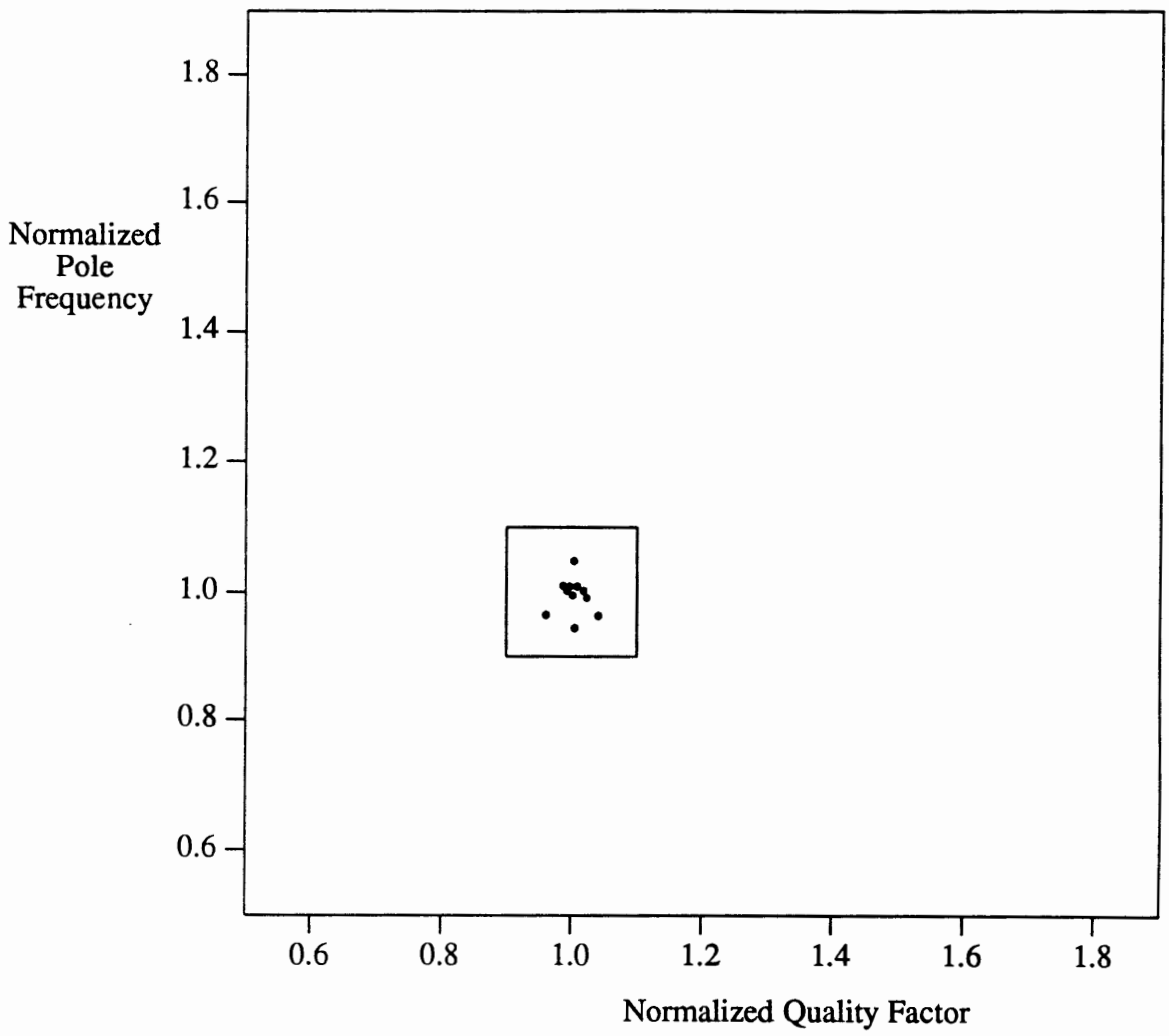
A.3. RESIDUAL ERRORS OF SELECTED SAMPLES



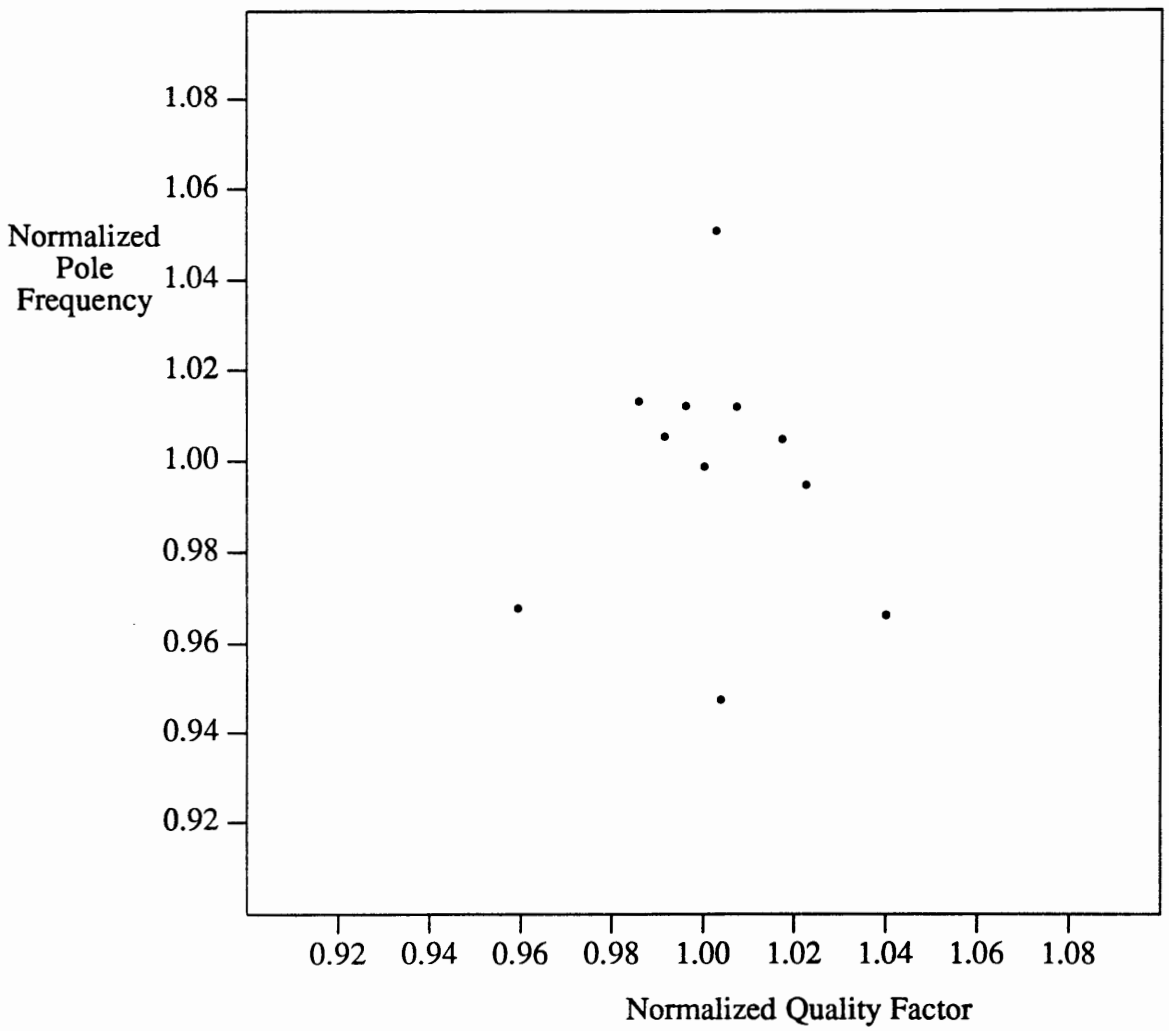
A19. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 3.33$, generalization data)



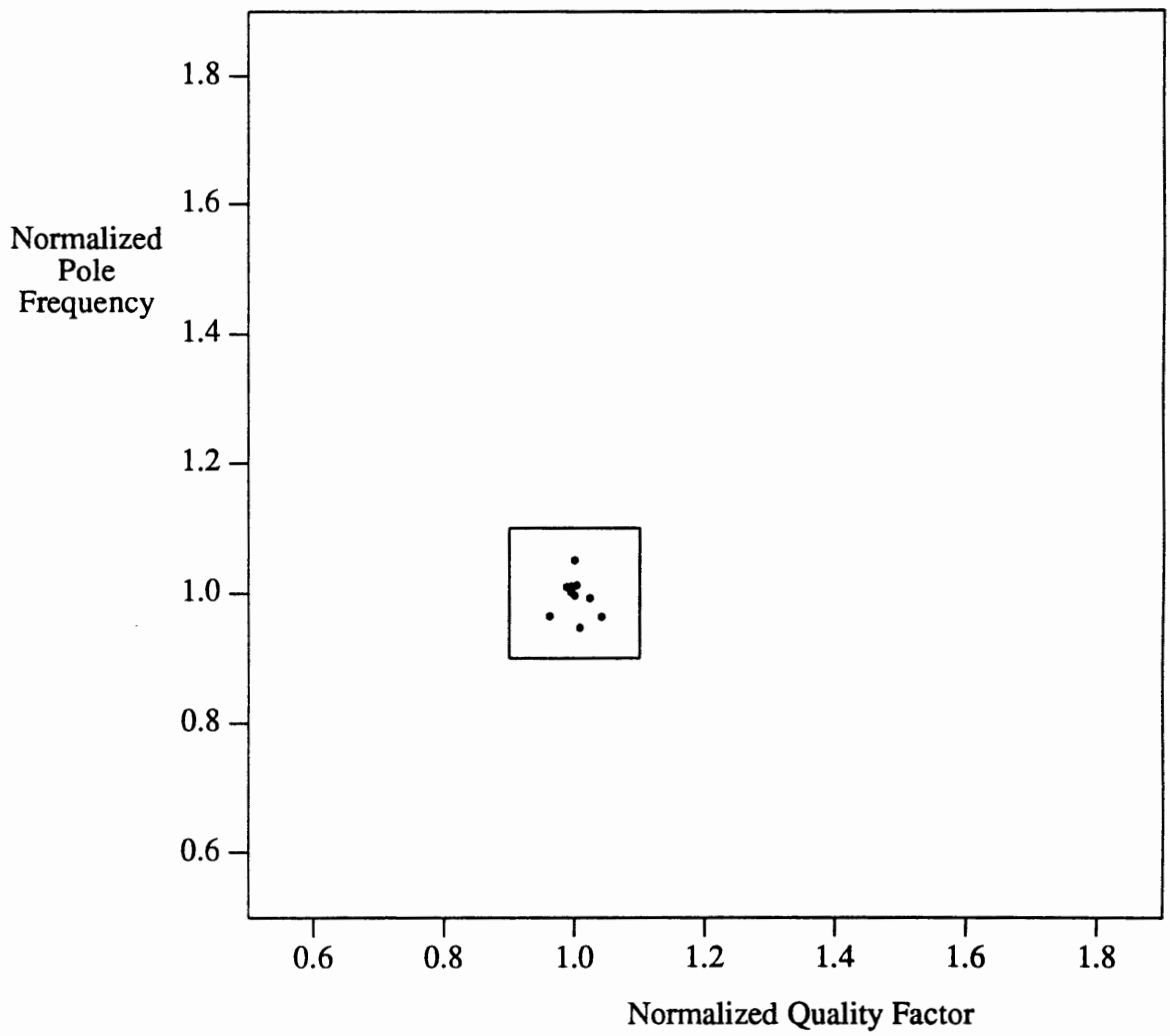
A20. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 3.33$, generalization data, enlargement of A19.)



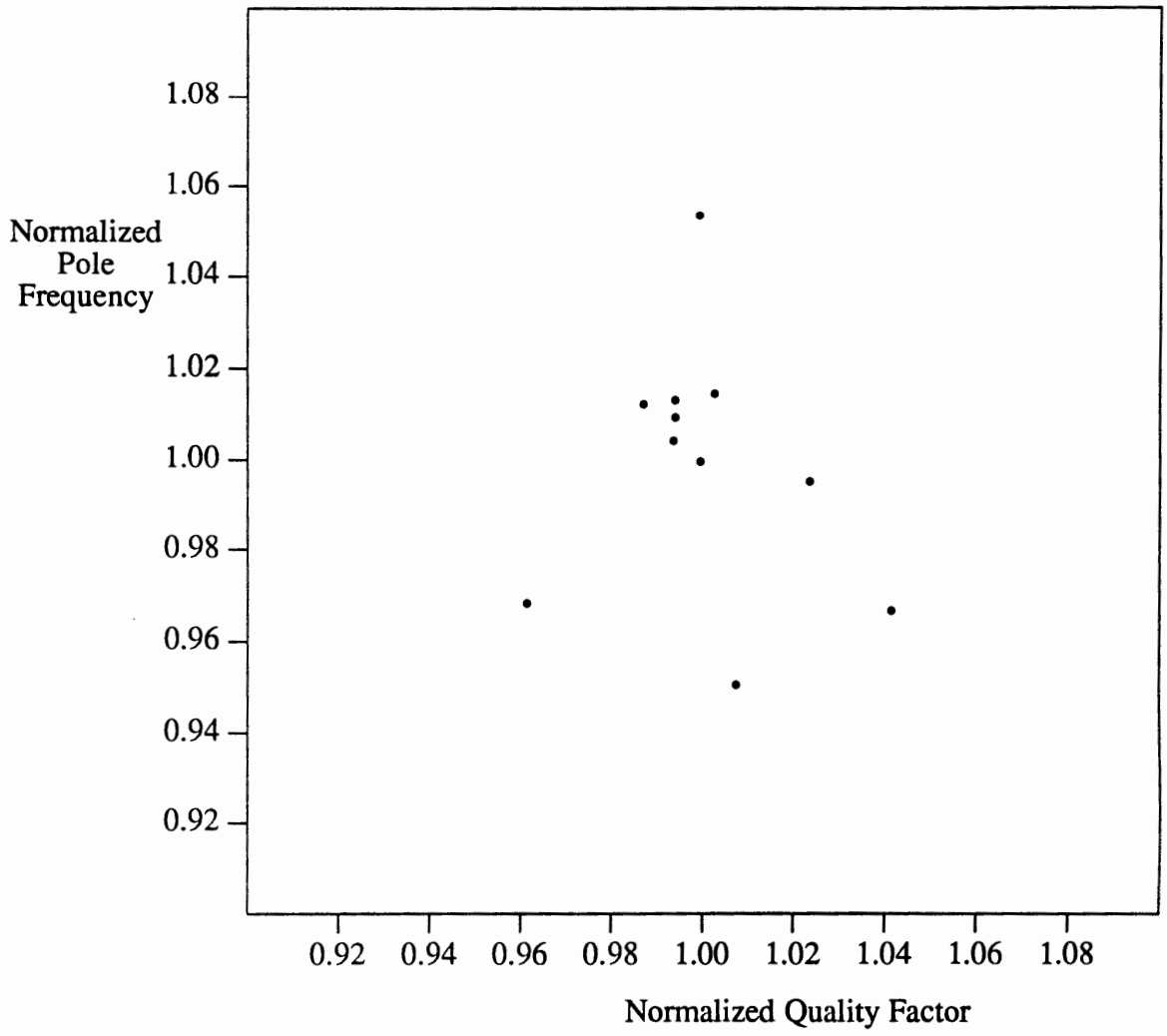
A21. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 2.5$, generalization data)



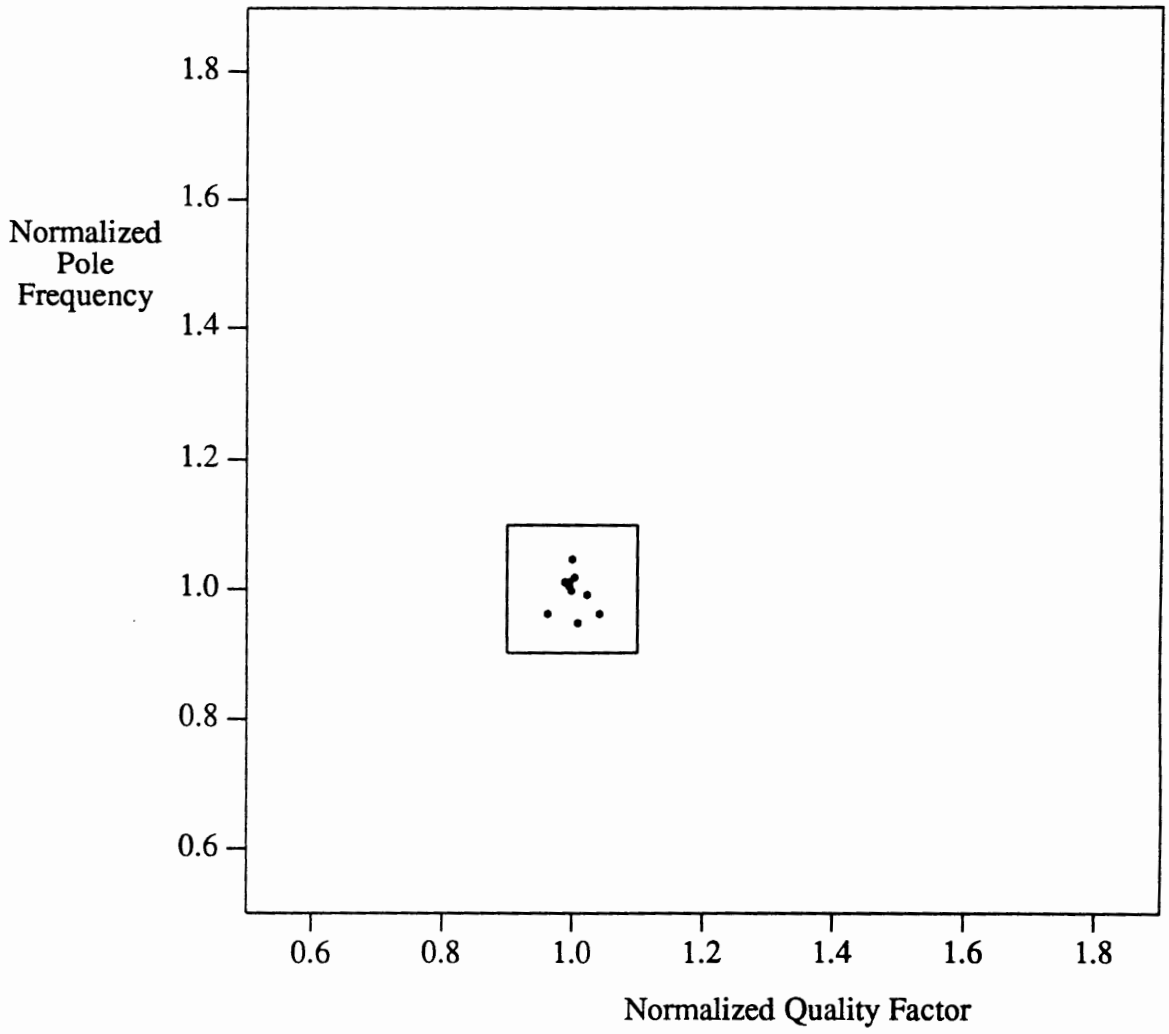
A22. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 2.5$, generalization data, enlargement of A21.)



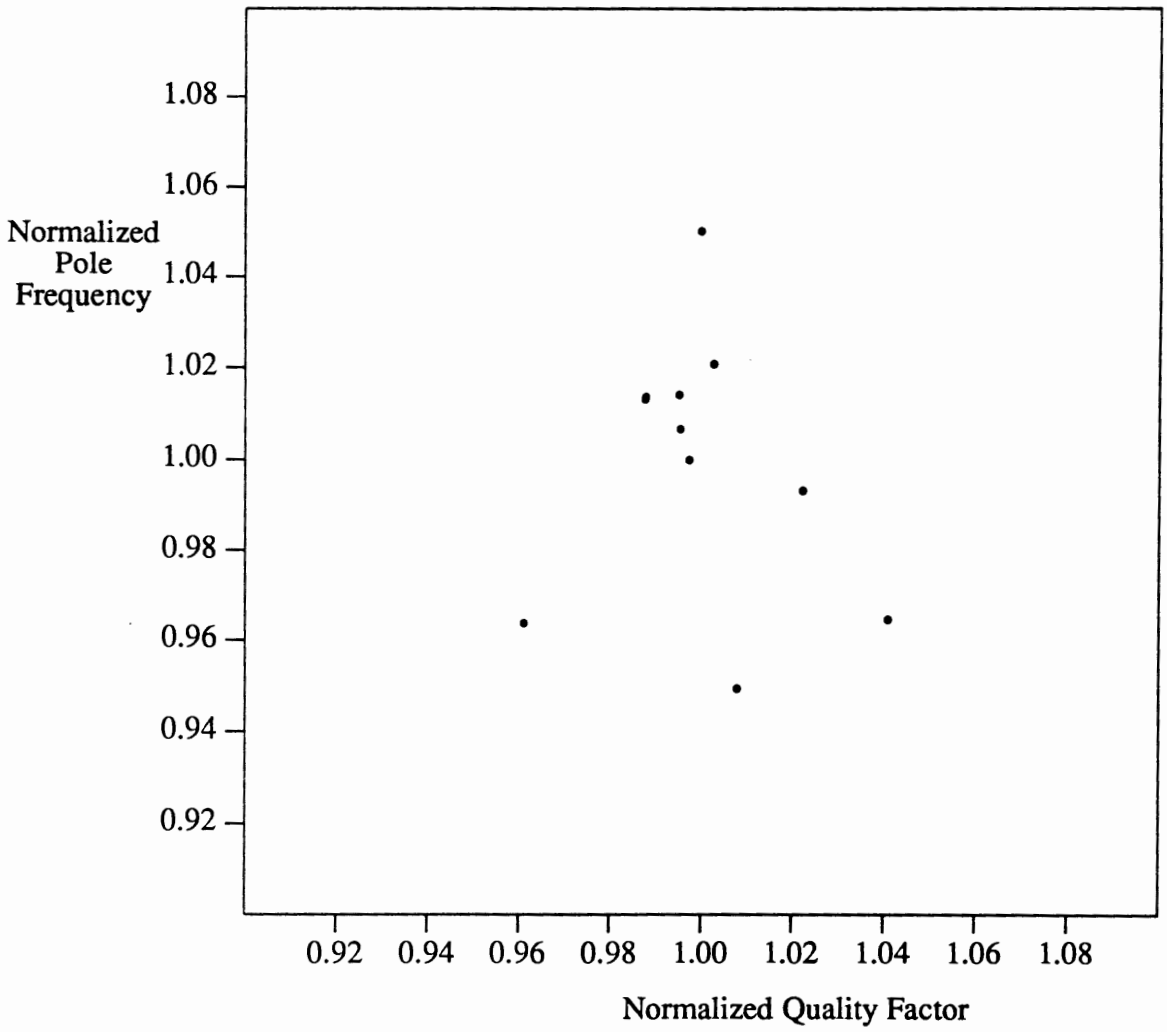
A23. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 1.67$, generalization data)



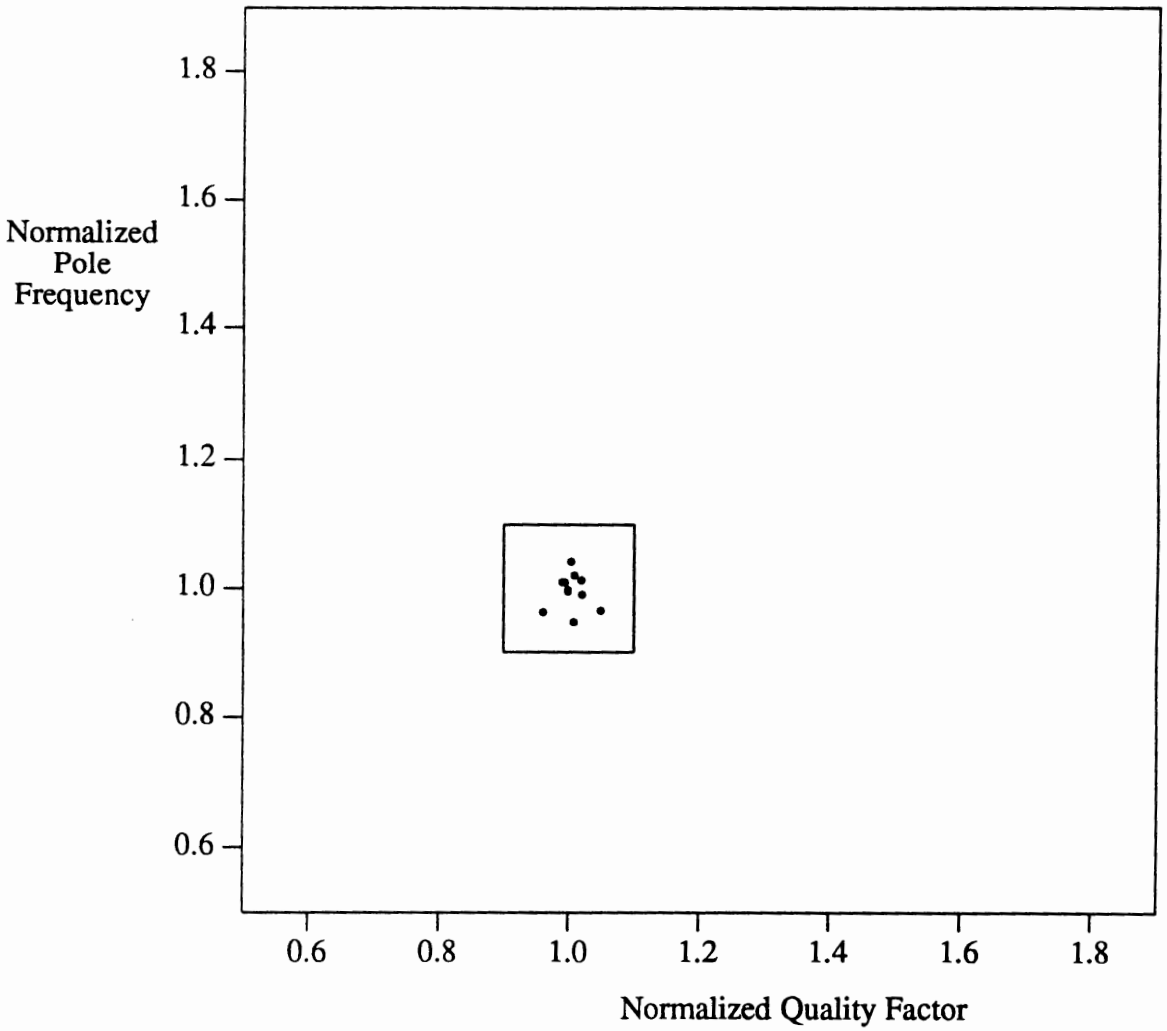
A24. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 1.67$, generalization data, enlargement of A23.)



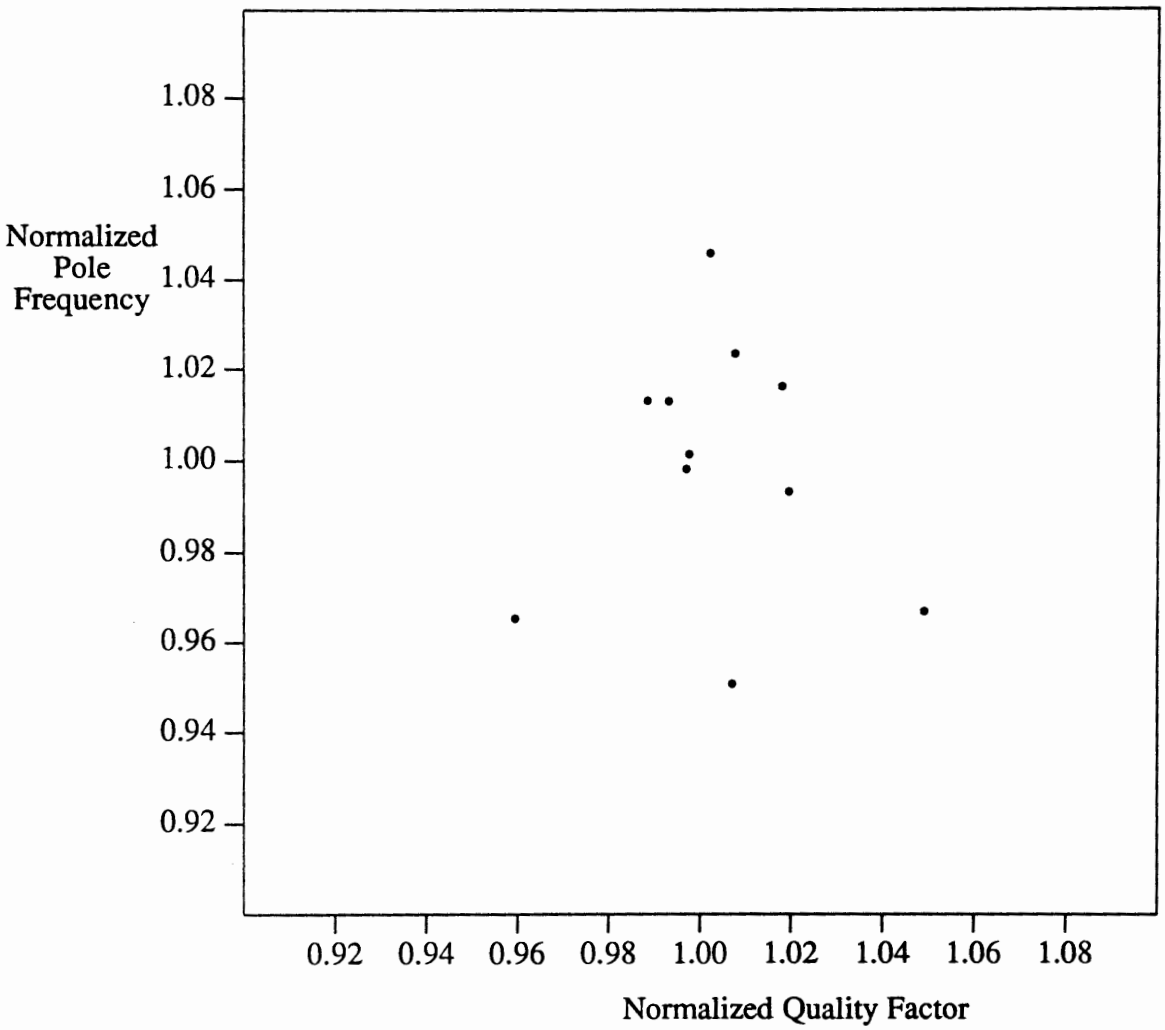
A25. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 1.25$, generalization data)



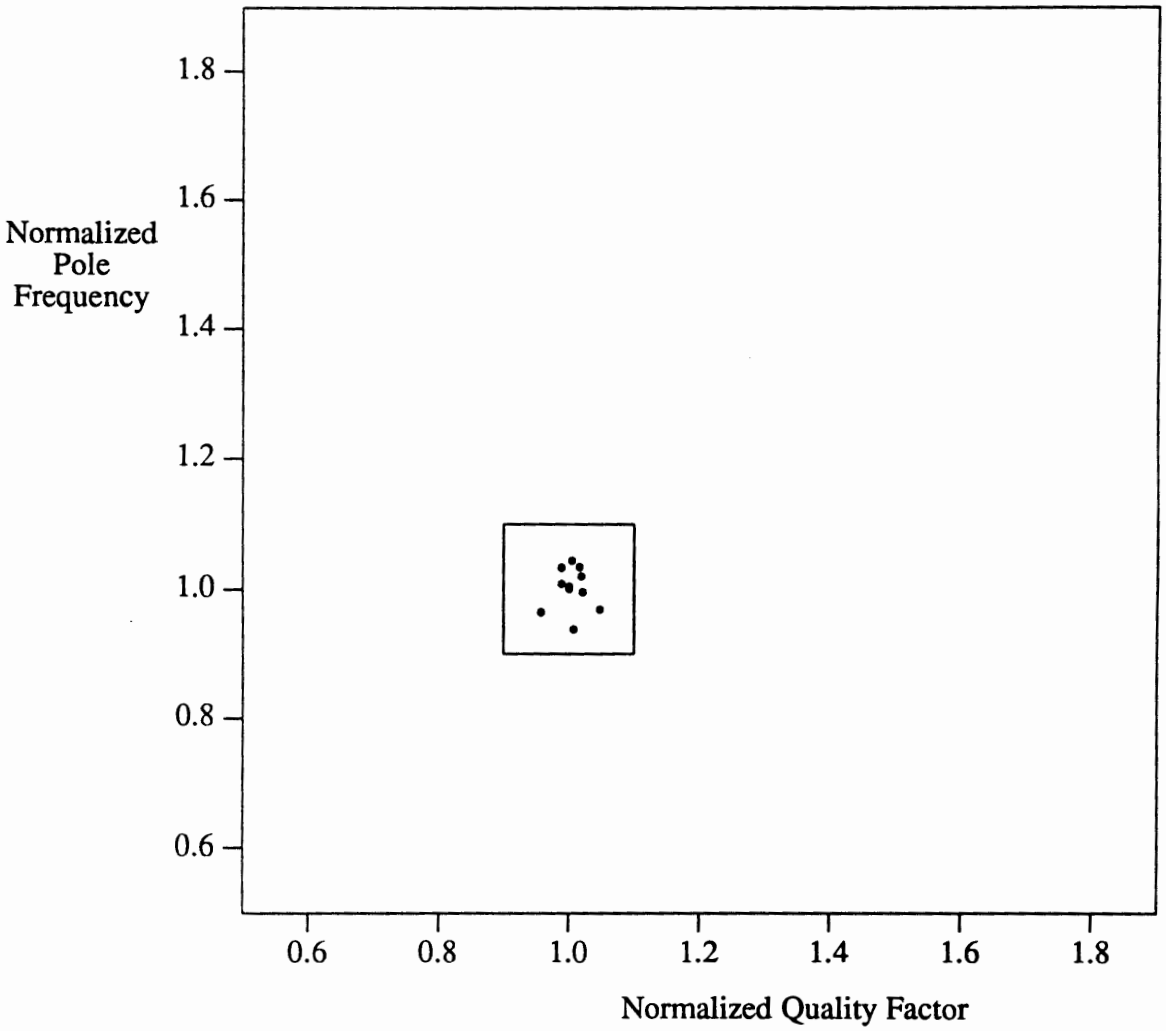
A26. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 1.25$, generalization data, enlargement of A25.)



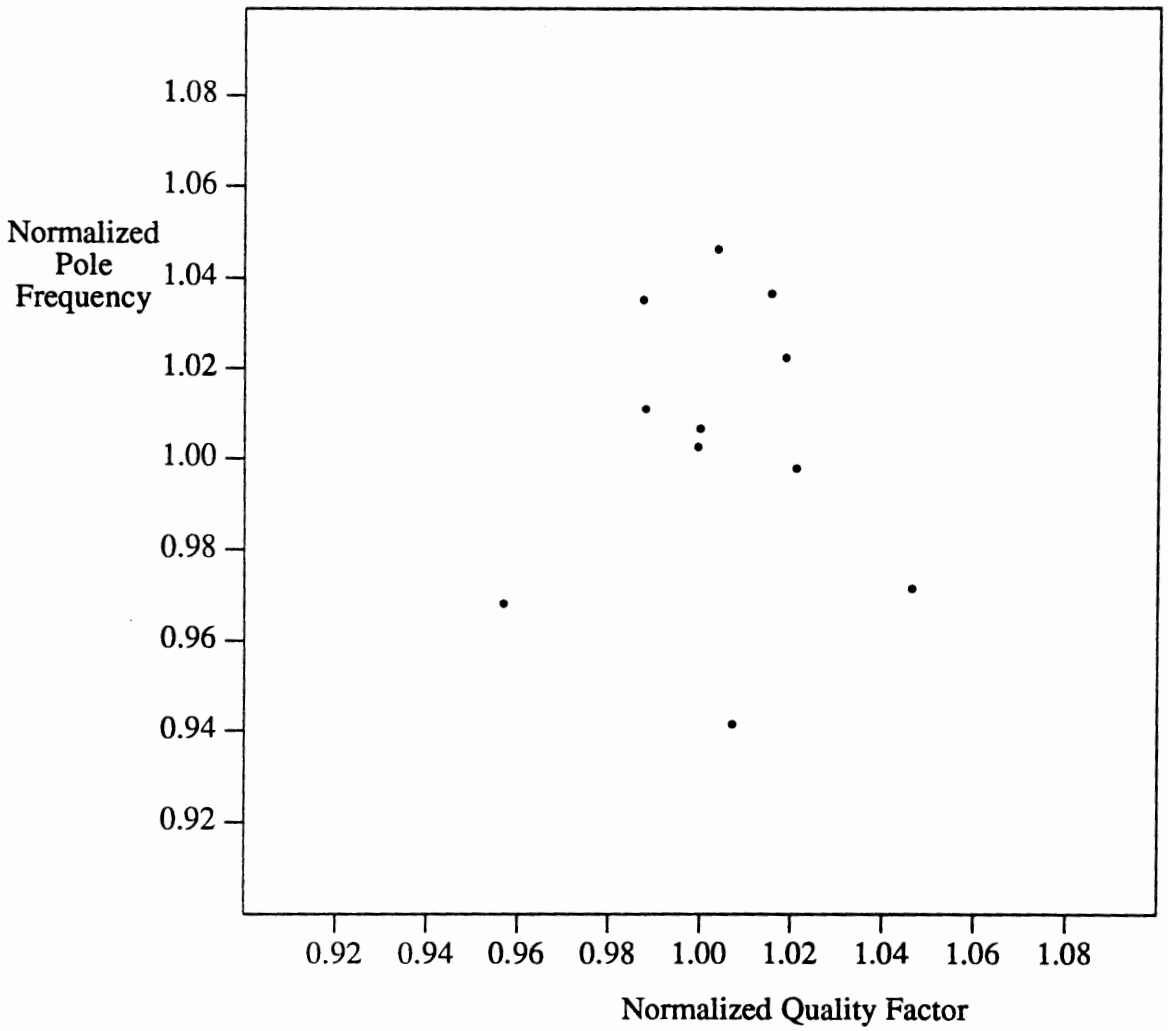
A27. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 1.0$, generalization data)



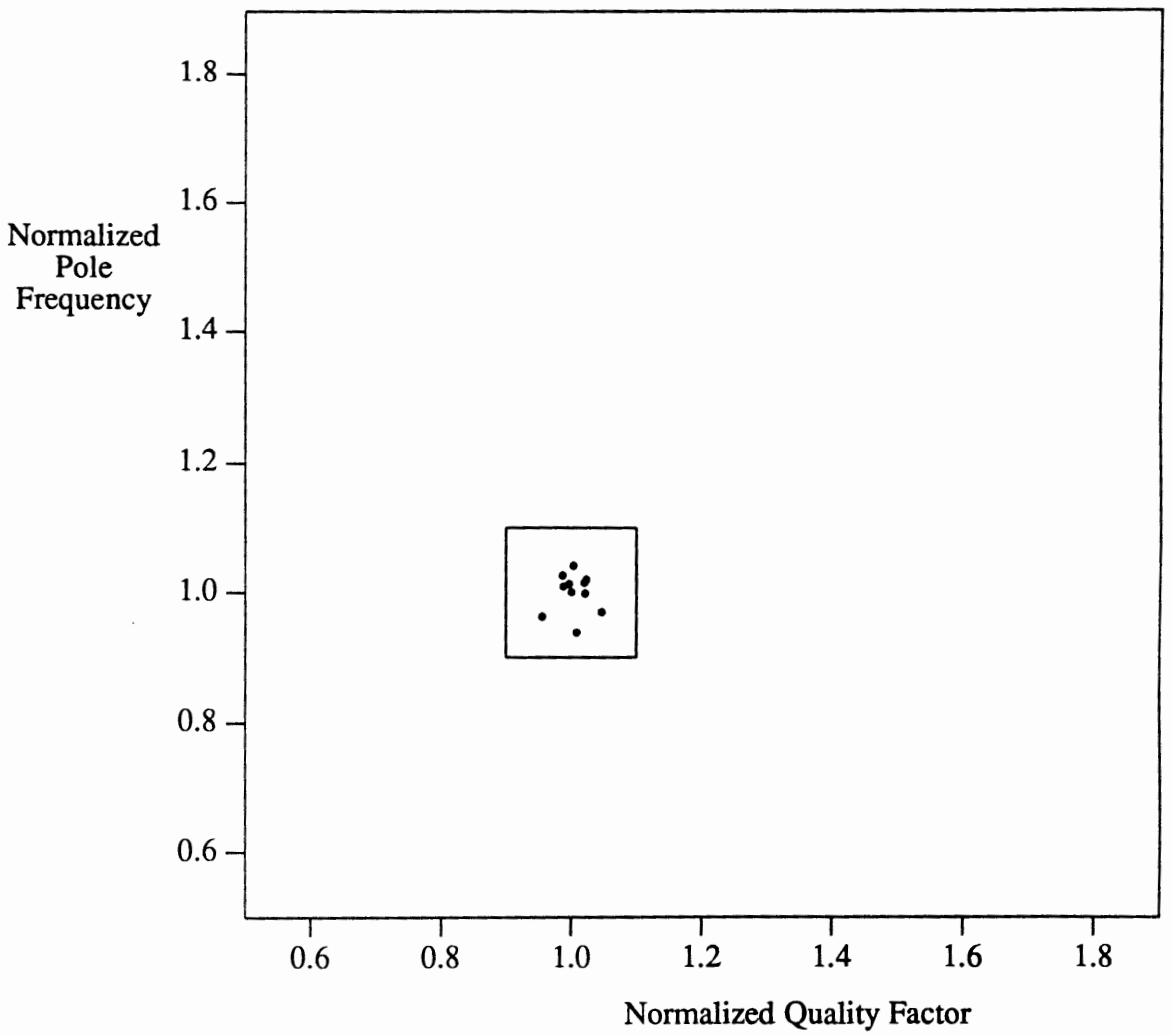
A28. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 1.0$, generalization data, enlargement of A27.)



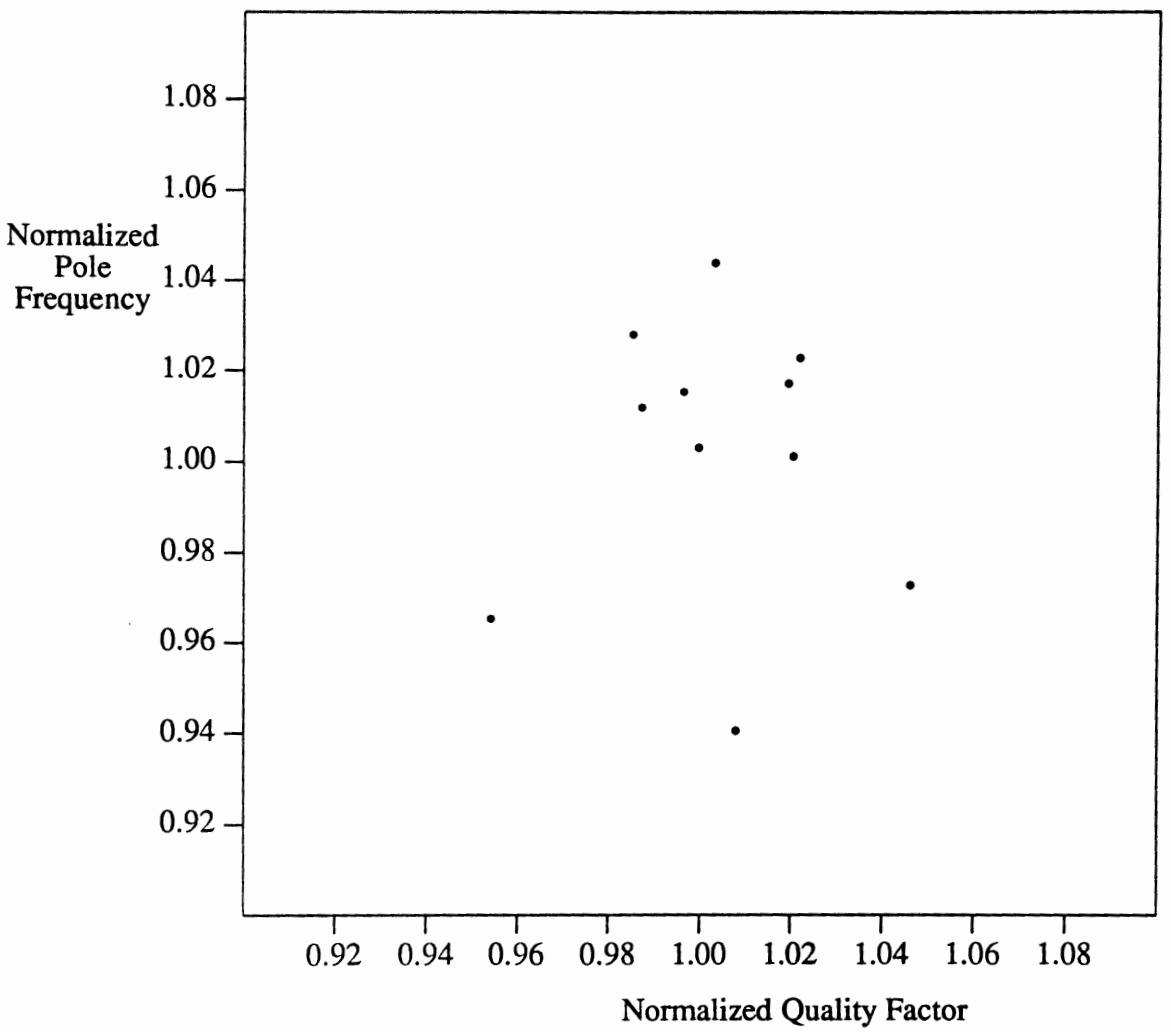
A29. Residual errors of Q_p and ω_p after tuning via NN
($Q_{p0} = 0.83$, generalization data)



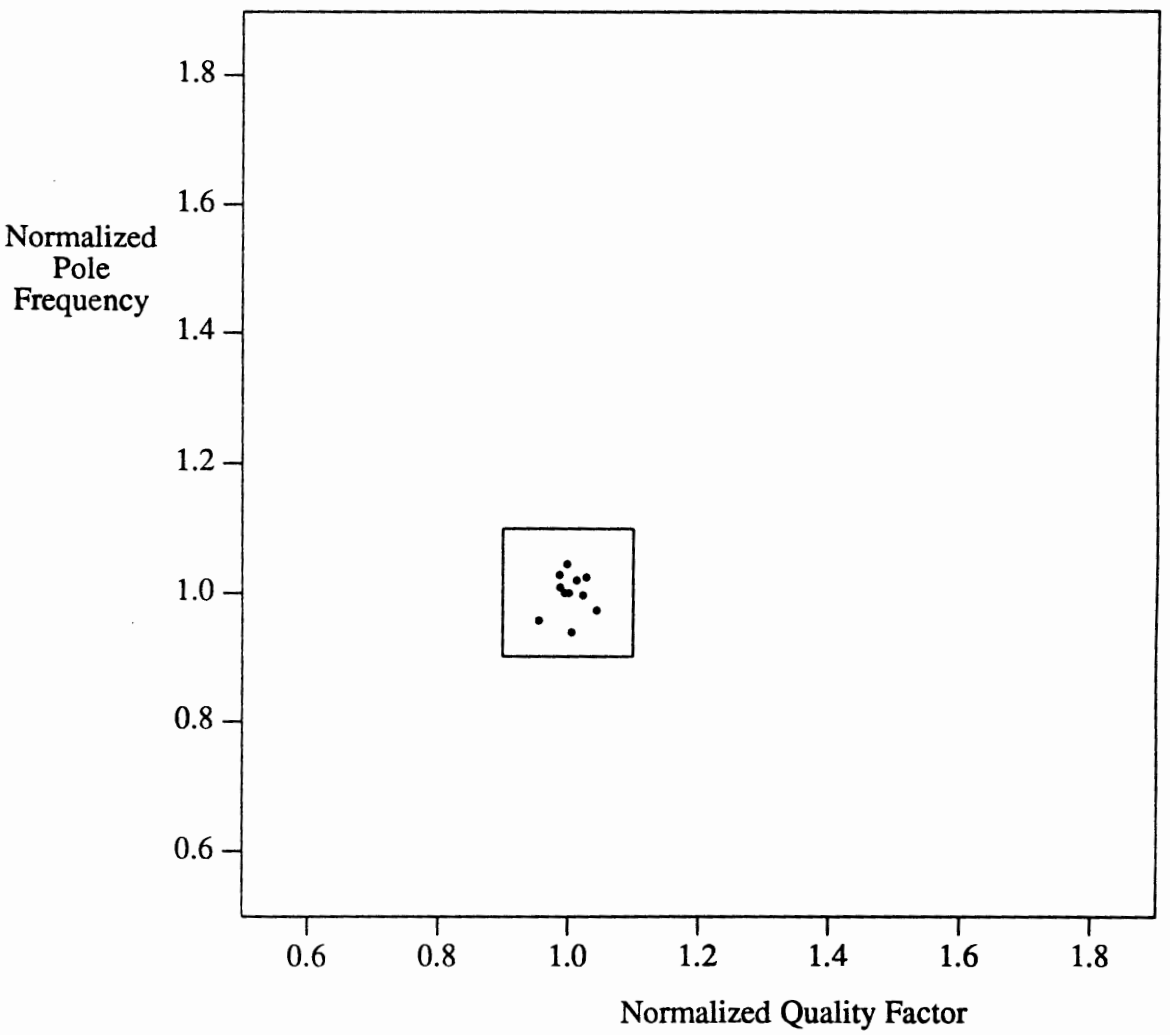
A30. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 0.83$, generalization data, enlargement of A29.)



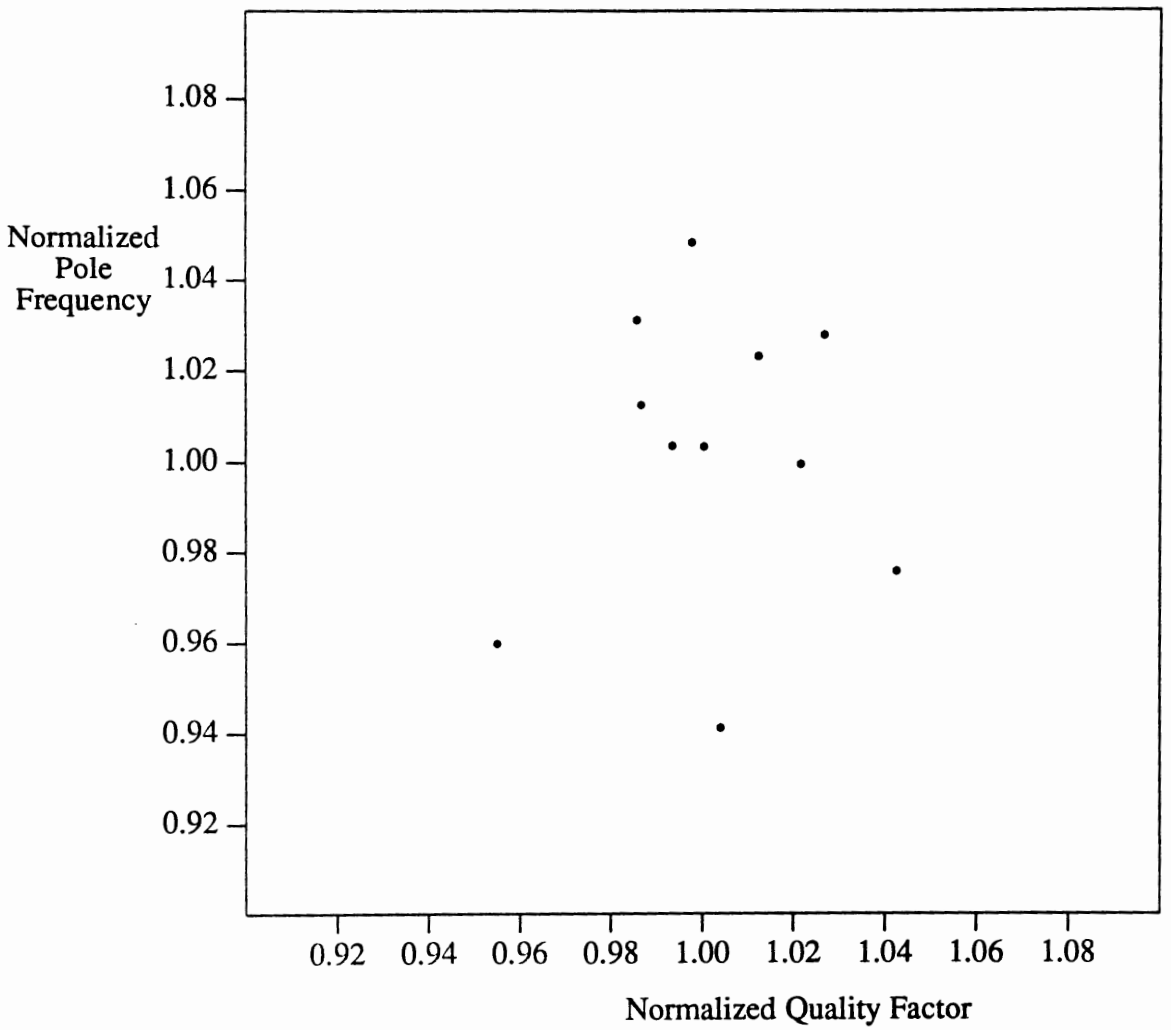
A31. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 0.707$, generalization data)



A32. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 0.707$, generalization data, enlargement of A31.)



A33. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 0.625$, generalization data)



A34. Residual errors of Q_p and ω_p after tuning via NN ($Q_{p0} = 0.625$, generalization data, enlargement of A33.)

APPENDIX B

MATLAB-FILES FOR DATA GENERATION

B.1. TRAINING RECORDS FOR RLC-FILTER

```
L0=1e-4;
C0=1e-10;
R0=2e+2;                          % quality factor 5

wo=sqrt(1/(L0*C0));               % pole frequency

dL=0.05*L0;
dC=0.05*C0;
dR=0.05*R0;
dw=0.05*wo;

wstart=0.15*wo;
Lstart=L0*(1-0.35);
Cstart=C0*(1-0.35);
Rstart=R0*(1-0.35);

n=0;
for l=1:13
   R=Rstart+l*dR;
   for m=1:13
      C=Cstart+m*dC;
      for k=1:13
          L=Lstart+k*dL;
          j=0;
          for q=1:7
            n=n+1;
              for p=1:7
              j=j+1;
              w=wstart+j*dw;
              c=w*w*L*C;
              co=w*w*L0*C0;
              b=2*(R*R*C/(L*2)-1);
              bo=2*(R0*R0*C0/(L0*2)-1);
              a=1+b*c+c*c;
              ao=1+bo*co+co*co;
              D=sqrt(a);
              D0=sqrt(ao);
              G(n,p)=1/D - 1/D0;
              end
          end
         n=n+1
          G(n,1)=R/R0;
          G(n,2)=(R-R0)/R0;
         G(n,3)=(C-C0)/C0;
         G(n,4)=(L-L0)/L0;
       end
   end
end
```

B.2. TESTING RECORDS FOR RLC-FILTER

```
L0=1e-4;
C0=1e-10;
R0=2e+2;                          % quality factor 5

wo=sqrt(1/(L0*C0));               % pole frequency

dL=0.05*L0;
dC=0.05*C0;
dR=0.05*R0;
dw=0.05*wo;

wstart=0.15*wo;
Lstart=L0*(1-0.325);
Cstart=C0*(1-0.325);
Rstart=R0*(1-0.325);              % in between points

n=0;
for l=1:12
   R=Rstart+l*dR;
   for m=1:12
      C=Cstart+m*dC;
      for k=1:12
          L=Lstart+k*dL;
          j=0;
          for q=1:7
            n=n+1;
              for p=1:7
              j=j+1;
              w=wstart+j*dw;
              c=w*w*L*C;
              co=w*w*L0*C0;
              b=2*(R*R*C/(L*2)-1);
              bo=2*(R0*R0*C0/(L0*2)-1);
              a=1+b*c+c*c;
              ao=1+bo*co+co*co;
              D=sqrt(a);
              D0=sqrt(ao);
              G(n,p)=1/D - 1/D0;
              end
          end
         n=n+1
          G(n,1)=R/R0;
          G(n,2)=(R-R0)/R0;
         G(n,3)=(C-C0)/C0;
         G(n,4)=(L-L0)/L0;
       end
   end
end
```

B.3. TRAINING RECORDS FOR g_m-C-FILTER

```
C10=12.5e-12; C20=0.5e-12;
g10=2.5e-5; g20=2.5e-5;                    % quality factor 5
w0=sqrt(g10*g20/(C10*C20));                % pole frequency
dC1=0.05*C10; dC2=0.05*C20;
dg1=0.05*g10; dg2=0.05*g20;
dw=0.05*w0; wstart=0.15*w0;
C1start=C10*(1-0.35); C2start=C20*(1-0.35);
g1start=g10*(1-0.35); g2start=g20*(1-0.35);

n=0;
for k=1:13 g1=g1start+k*dg1;
   for j=1:13 g2=g2start+j*dg2;
      for l=1:13                           % C1/C2 stays constant within 0.1% !
         C1=C1start+l*dC1;
         C3(1)=C2start+l*dC2;
         C3(2)=C2start+(l-1)*dC2;
         C3(3)=C2start+(l+1)*dC2;
         if l > 1
         if l < 13
         for m=1:3
            C2=C3(m);
            i=0;
            for q=1:7
            n=n+1;
                  for p=1:7 i=i+1;
                  w=wstart+i*dw;
                  c=w*w*C1*C2/(g1*g2);
                  co=w*w*C10*C20/(g10*g20);
                  b=2*(g2*C2/(2*g1*C1)-1);
                  bo=2*(g20*C20/(2*g10*C10)-1);
                  a=1+b*c+c*c;
                  ao=1+bo*co+co*co;
                  D=sqrt(a);
                  D0=sqrt(ao);
                  G(n,p)=1/D - 1/D0;
                  end
            end
            n=n+1
            G(n,1)=(C2-C20)/C20-(g1-g10)/g10;
            G(n,2)=(C1-C10)/C10-(g2-g20)/g20;
            G(n,3)=C1/C10;                 % term B
            G(n,4)=C2/C20;                 % term A
         end
         end                               % end of if
         end                               % end of if
         if l == 1
         C3(2)=C3(3);
         for m=1:2
            C2=C3(m);
```

```
            i=0;
            for q=1:7
            n=n+1;
                  for p=1:7 i=i+1;
                  w=wstart+i*dw;
                  c=w*w*C1*C2/(g1*g2);
                  co=w*w*C10*C20/(g10*g20);
                  b=2*(g2*C2/(2*g1*C1)-1);
                  bo=2*(g20*C20/(2*g10*C10)-1);
                  a=1+b*c+c*c;
                  ao=1+bo*co+co*co;
                  D=sqrt(a);
                  D0=sqrt(ao);
                  G(n,p)=1/D - 1/D0;
                  end
            end
            n=n+1
            G(n,1)=(C2-C20)/C20-(g1-g10)/g10;
            G(n,2)=(C1-C10)/C10-(g2-g20)/g20;
            G(n,3)=C1/C10;                        % term B
            G(n,4)=C2/C20;                        % term A
         end
         end                                      % end of if
         if l == 13
         for m=1:2
            C2=C3(m);
            i=0;
            for q=1:7
            n=n+1;
                  for p=1:7 i=i+1;
                  w=wstart+i*dw;
                  c=w*w*C1*C2/(g1*g2);
                  co=w*w*C10*C20/(g10*g20);
                  b=2*(g2*C2/(2*g1*C1)-1);
                  bo=2*(g20*C20/(2*g10*C10)-1);
                  a=1+b*c+c*c;
                  ao=1+bo*co+co*co;
                  D=sqrt(a);
                  D0=sqrt(ao);
                  G(n,p)=1/D - 1/D0;
                  end
            end
            n=n+1
            G(n,1)=(C2-C20)/C20-(g1-g10)/g10;
            G(n,2)=(C1-C10)/C10-(g2-g20)/g20;
            G(n,3)=C1/C10;                        % term B
            G(n,4)=C2/C20;                        % term A
         end
         end                                      % end of if
      end
   end
end
```

B.4. TESTING RECORDS FOR g_m-C-FILTER

```
C10=12.5e-12; C20=0.5e-12;
g10=2.5e-5; g20=2.5e-5;                          % quality factor 5
w0=sqrt(g10*g20/(C10*C20));                      % pole frequency
dC1=0.05*C10; dC2=0.05*C20;
dg1=0.05*g10; dg2=0.05*g20;
dw=0.05*w0; wstart=0.15*w0;
C1start=C10*(1-0.325); C2start=C20*(1-0.325);
g1start=g10*(1-0.325); g2start=g20*(1-0.325);    % in between points

n=0;
for k=1:12 g1=g1start+k*dg1;
   for j=1:12 g2=g2start+j*dg2;
       for l=1:12
          C1=C1start+l*dC1;
          C3(1)=C2start+l*dC2;
          C3(2)=C2start+(l-1)*dC2;
          C3(3)=C2start+(l+1)*dC2;
          if l > 1
          if l < 12
          for m=1:3
              C2=C3(m);
              i=0;
              for q=1:7
              n=n+1;
                    for p=1:7 i=i+1;
                    w=wstart+i*dw;
                    c=w*w*C1*C2/(g1*g2);
                    co=w*w*C10*C20/(g10*g20);
                    b=2*(g2*C2/(2*g1*C1)-1);
                    bo=2*(g20*C20/(2*g10*C10)-1);
                    a=1+b*c+c*c;
                    ao=1+bo*co+co*co;
                    D=sqrt(a);
                    D0=sqrt(ao);
                    G(n,p)=1/D - 1/D0;
                    end
              end
              n=n+1
              G(n,1)=(C2-C20)/C20-(g1-g10)/g10;
              G(n,2)=(C1-C10)/C10-(g2-g20)/g20;
              G(n,3)=C1/C10;                     % term B
              G(n,4)=C2/C20;                     % term A
          end
          end                                    % end of if
          end                                    % end of if
          if l == 1
          C3(2)=C3(3);
          for m=1:2
              C2=C3(m);
```

```
            i=0;
            for q=1:7
            n=n+1;
                  for p=1:7 i=i+1;
                  w=wstart+i*dw;
                  c=w*w*C1*C2/(g1*g2);
                  co=w*w*C10*C20/(g10*g20);
                  b=2*(g2*C2/(2*g1*C1)-1);
                  bo=2*(g20*C20/(2*g10*C10)-1);
                  a=1+b*c+c*c;
                  ao=1+bo*co+co*co;
                  D=sqrt(a);
                  D0=sqrt(ao);
                  G(n,p)=1/D - 1/D0;
                  end
            end
            n=n+1
            G(n,1)=(C2-C20)/C20-(g1-g10)/g10;
            G(n,2)=(C1-C10)/C10-(g2-g20)/g20;
            G(n,3)=C1/C10;                          % term B
            G(n,4)=C2/C20;                          % term A
         end
         end                                        % end of if
         if l == 12
         for m=1:2
            C2=C3(m);
            i=0;
            for q=1:7
            n=n+1;
                  for p=1:7 i=i+1;
                  w=wstart+i*dw;
                  c=w*w*C1*C2/(g1*g2);
                  co=w*w*C10*C20/(g10*g20);
                  b=2*(g2*C2/(2*g1*C1)-1);
                  bo=2*(g20*C20/(2*g10*C10)-1);
                  a=1+b*c+c*c;
                  ao=1+bo*co+co*co;
                  D=sqrt(a);
                  D0=sqrt(ao);
                  G(n,p)=1/D - 1/D0;
                  end
            end
            n=n+1
            G(n,1)=(C2-C20)/C20-(g1-g10)/g10;
            G(n,2)=(C1-C10)/C10-(g2-g20)/g20;
            G(n,3)=C1/C10;                          % term B
            G(n,4)=C2/C20;                          % term A
         end
         end                                        % end of if
      end
   end
end
```

B.5. TESTING RECORDS (2nd ITERATION) FOR g_m-C-FILTER

```
Qn=5;                          % nominal quality factor
wn=1;
dw=0.05*wn;
wstart=0.15*wn;

A=e18gen;                  % load .nnr file !!
B=qwgen11;                     % load qw-actual-data-file

C1(1)=-0.275;                  % keep track of the delta-Cs
C1(2)=0.025;
C1(3)=0.275;
C1(4)=0.125;
C1(5)=-0.275;
C1(6)=0.125;
C1(7)=0.225;
C1(8)=-0.125;
C1(9)=0.075;
C1(10)=-0.225;
C1(11)=-0.275;

C2(1)=-0.275;
C2(2)=0.025;
C2(3)=0.275;
C2(4)=0.125;
C2(5)=-0.225;
C2(6)=0.125;
C2(7)=0.225;
C2(8)=-0.125;
C2(9)=0.025;
C2(10)=-0.275;
C2(11)=-0.275;

help(1)=1;                 % record numbers of test points
help(2)=15;
help(3)=33;
help(4)=206;
help(5)=376;
help(6)=398;
help(7)=1803;
help(8)=2044;
help(9)=2470;
help(10)=4492;
help(11)=4863;

for k=1:11
   g1(k)=A(help(k),1)-A(help(k),3);  % tuning amount for 2nd iteration:
   g2(k)=A(help(k),2)-A(help(k),4);  % calculated-measured NN output
end                            % (of 1st iteration)
```

```
for k=1:11
   AB(k,1)=1+C1(k);                    % new term B
   AB(k,2)=1+C2(k);                    % new term A
end

n=0;
for k=1:11
   wa=B(k,2)*wn;
   Qa=B(k,1)*Qn;
   i=0;
       for q=1:7
          n=n+1;
          for p=1:7
              i=i+1;
              w=wstart+i*dw;
              a=w*w/(wa*wa);
              a0=w*w/(wn*wn);
              b=1/(Qa*Qa)-2;
              b0=1/(Qn*Qn)-2;
              c=sqrt(1+b*a+a*a);
              c0=sqrt(1+b0*a0+a0*a0);
              G(n,p)=1/c-1/c0;
          end
       end
   n=n+1
   G(n,1)=g1(k);
   G(n,2)=g2(k);                       % -->.nna-file
end
end;
```

APPENDIX C

MATLAB- AND C-FILES FOR EVALUATION

C.1. CALCULATION OF THE EUCLEDIAN DISTANCE FOR CIRCUIT A

```
#include <stdio.h>
#include <math.h>
#define FILE_LENGTH 2197                                % Number of records
float  R0, C0, L0, R, L, C, Rmin, Rmax, Cmin, Cmax, Lmin, Lmax;
float  errorR,errorC,errorL,delta,min,max,help,average;
float  min,max,help;
int j;

main()
{
      min=1, max=0, help=0;
      Rmin=1, Rmax=0, Cmin=1, Cmax=0, Lmin=1, Lmax=0;
      printf("0esults for the training data0);
      for(j=1; j<(FILE_LENGTH + 1); j++)
      {
            scanf("%f %f %f %f %f %f", &R0, &C0, &L0, &R, &C, &L);

            errorR=R-R0;
            errorC=C-C0;
            errorL=L-L0;

            if (errorR<Rmin) Rmin=errorR;
            if (errorC<Cmin) Cmin=errorC;
            if (errorL<Lmin) Lmin=errorL;

            if (errorR>Rmax) Rmax=errorR;
            if (errorC>Cmax) Cmax=errorC;
            if (errorL>Lmax) Lmax=errorL;

            delta=sqrt(errorR*errorR+errorC*errorC+errorL*errorL);

            if (delta<min) min=delta;
            if (delta>max) max=delta;

            average=help+delta;
            help=average;
      }
average=help/FILE_LENGTH;
printf ("average: %4.6f0,average);
printf ("min: %4.6f0,min);
printf ("max: %4.6f0,max);
printf ("Rmin: %4.6f0,Rmin);
printf ("Rmax: %4.6f0,Rmax);
printf ("Cmin: %4.6f0,Cmin);
printf ("Cmax: %4.6f0,Cmax);
printf ("Lmin: %4.6f0,Lmin);
printf ("Lmax: %4.6f0,Lmax);
}
```

C.2. INITIAL VALUES OF THE QUALITY FACTOR Q_p and THE POLE FREQUENCY ω_p

```
C10=12.5e-12; C20=0.5e-12;
g10=2.5e-5; g20=g10;
w0=sqrt(g10*g20/(C10*C20));                 % nominal pole frequency
q0=sqrt(g10*C10/(g20*C20));                 % nominal quality factor
dC1=0.05*C10; dC2=0.05*C20;
dg1=0.05*g10; dg2=0.05*g20;
dw=0.05*w0; wstart=0.15*w0;
C1start=C10*(1-0.35); C2start=C20*(1-0.35);
g1start=g10*(1-0.35); g2start=g20*(1-0.35);

n=0;
for k=1:13 g1=g1start+k*dg1;
   for j=1:13 g2=g2start+j*dg2;
       for l=1:13
          C1=C1start+l*dC1;
          C3(1)=C2start+l*dC2;
          C3(2)=C2start+(l-1)*dC2;
          C3(3)=C2start+(l+1)*dC2;
          if l > 1
          if l < 13
          for m=1:3
              C2=C3(m);
              n=n+1
              G(n,1)=sqrt(g1*C1/(g2*C2))/q0;    % normalized quality factor
              G(n,2)=sqrt(g1*g2/(C1*C2))/w0;    % normalized pole frequency
          end
          end                                   % end of if
          end                                   % end of if
          if l == 1
          C3(2)=C3(3);
          for m=1:2
              C2=C3(m);
              n=n+1
              G(n,1)=sqrt(g1*C1/(g2*C2))/q0;    % normalized quality factor
              G(n,2)=sqrt(g1*g2/(C1*C2))/w0;    % normalized pole frequency
          end
          end                                   % end of if
          if l == 13
          for m=1:2
              C2=C3(m);
              n=n+1
              G(n,1)=sqrt(g1*C1/(g2*C2))/q0;    % normalized quality factor
              G(n,2)=sqrt(g1*g2/(C1*C2))/w0;    % normalized pole frequency
          endend                                % end of if
      end
   end
end
```

C.3. ACTUAL QUALITY FACTOR Q_a AND POLE FREQUENCY ω_a AFTER TUNING

```
A=AB;                                   % load the error-terms A and B !!
B=e18;                                  % load the .nnr result-file
int1=0;
int2=0;
for n=1:6253                            % number of records
a1(n)=(B(n,3)-B(n,1))/A(n,2);           % e1/A
a2(n)=(B(n,4)-B(n,2))/A(n,1);           % e2/B
output(n,1)=sqrt((1+a1(n))/(1+a2(n)));          % Q actual
output(n,2)=sqrt(1+a1(n)+a2(n)+a1(n)*a2(n));    % w actual
if output(n,1) > 0.95
      if output(n,1) < 1.05
            if output(n,2) > 0.95
                  if output(n,2) < 1.05
                  int1=int1+1;
                  end
            end
      end
end
if output(n,1) > 0.99
      if output(n,1) < 1.01
            if output(n,2) > 0.99
                  if output(n,2) < 1.01
                  int2=int2+1;
                  end
            end
      end
end
end
qmin=min(output(:,1))                   % minimum quality factor
qmax=max(output(:,1))                   % maximum quality factor
wmin=min(output(:,2))                   % minimum pole frequency
wmax=max(output(:,2))                   % maximum pole frequency
range1=int1                       % number of records in 5% intervall
range2=int2                       % number of records in 1% intervall
end;
```

APPENDIX D

NETWORK PARAMETERS

D.1. INITIAL NETWORK FOR CIRCUIT A

```
Title: Q=5.0/50in/18hid/3out
     Display Mode: Network                    Type: Hetero-Associative
    Display Style: default
  Control Strategy: backprop          L/R Schedule: backprop
        0 Learn          0 Recall          0 Layer
        1 Aux 1          0 Aux 2           0 Aux 3
L/R Schedule: backprop
  Recall Step          1       0       0       0       0
    Input Clamp      0.0000  0.0000  0.0000  0.0000  0.0000
    Firing Density 100.0000  0.0000  0.0000  0.0000  0.0000
    Temperature      0.0000  0.0000  0.0000  0.0000  0.0000
    Gain           1.0000  0.0000  0.0000  0.0000  0.0000
    Gain           1.0000  0.0000  0.0000  0.0000  0.0000
  Learn Step          5000       0       0       0       0
    Coefficient 1    0.9000  0.0000  0.0000  0.0000  0.0000
    Coefficient 2    0.6000  0.0000  0.0000  0.0000  0.0000
    Coefficient 3    0.0000  0.0000  0.0000  0.0000  0.0000
    Temperature      0.0000  0.0000  0.0000  0.0000  0.0000
IO Parameters
      Learn Data: File Rand. (einsR) Binary Load
     Recall Data: File Seq. (einsR)
     Result File: Desired Output, Output
   UserIO Program: userio
     I/P Ranges:  -1.0000,   1.0000
     O/P Ranges:  -0.3000,   0.3000
    I/P Start Col:      1          MinMax Table: einsr
    O/P Start Col:     51          Number of Entries: 56
MinMax Table <einsr>:
Col:      1       2       3       4       5       6
Min:  -0.0832 -0.1193 -0.1557 -0.1903 -0.2214 -0.2481
Max:   0.0384  0.0602  0.0869    .121   .1787   .2626
Col:      7       8       9      10      11      12
Min:  -0.2697 -0.2861 -0.3321 -0.4421 -0.5930 -0.8055
Max:    .3889   .5881    .924   1.542   2.783   4.727
Col:     13      14      15      16      17      18
Min:  -1.1132 -1.5674 -2.2207 -3.0582 -3.6985 -3.2079
Max:   4.902  4.882  4.533  4.16   4.265  5.434
Col:     19      20      21      22      23      24
Min:  -2.3863 -1.7522 -1.3240 -1.0343 -0.8319 -0.6856
Max:   5.883  6.234  6.489  6.473  6.486  6.313
Col:     25      26      27      28      29      30
Min:  -0.5763 -0.4926 -0.4268 -0.3741 -0.3312 -0.2958
Max:   6.131  6.006  4.802  3.588  2.718  2.125
Col:     31      32      33      34      35      36
Min:  -0.2661 -0.2409 -0.2194 -0.2008 -0.1846 -0.1705
Max:    1.71   1.412  1.189  1.019   .8849   .7776
Col:     37      38      39      40      41      42
Min:  -0.1580 -0.1469 -0.1370 -0.1282 -0.1202 -0.1130
Max:    .6902   .6178   .5571   .5056   .4615   .4234
Col:     43      44      45      46      47      48
```

```
Min:   -0.1064    -0.1005    -0.0950    -0.0900    -0.0854    -0.0812
Max:    .3901      .361      .3352      .3123      .2918      .2735
Col:      49        50         51         52         53         54
Min:   -0.0773    -1.3000    -0.3000    -0.3000    -0.3000     0.0000
Max:    .2569      1.3        .3         .3         .3          0
Col:      55        56
Min:    0.0000    0.0000
Max:       0         0
Layer: 1
     PEs: 1     Wgt Fields: 2              Sum: Sum
  Spacing: 5     F' offset: 0.00         Transfer: Linear
    Shape: Square                      Output: Direct
    Scale: 1.00   Low Limit: -9999.00   Error Func: standard
   Offset: 0.00   High Limit: 9999.00       Learn: --None--
 Init Low: -0.100  Init High: 0.100    L/R Schedule: (Network)
 Winner 1: None                     Winner 2: None
 PE: Bias
    1.000 Err Factor     0.000 Desired
    0.000 Sum          1.000 Transfer       1.000 Output
       0 Weights       -6.664 Error         0.000 Current Error
Layer: In
     PEs: 50    Wgt Fields: 1              Sum: Sum
  Spacing: 5     F' offset: 0.00         Transfer: Linear
    Shape: Square                      Output: Direct
    Scale: 1.00   Low Limit: -9999.00   Error Func: standard
   Offset: 0.00   High Limit: 9999.00       Learn: --None--
 Init Low: -0.100  Init High: 0.100    L/R Schedule: (Network)
 Winner 1: None                     Winner 2: None
 PE: 2
    1.000 Err Factor     0.645 Desired
    0.000 Sum          0.000 Transfer       0.000 Output
       0 Weights       0.000 Error          0.000 Current Error
 PE: 3
    1.000 Err Factor     0.619 Desired
    0.000 Sum          0.000 Transfer       0.000 Output
       0 Weights       0.000 Error          0.000 Current Error
 PE: 4
    1.000 Err Factor     0.589 Desired
    0.000 Sum          0.000 Transfer       0.000 Output
       0 Weights       0.000 Error          0.000 Current Error
 PE: 5
    1.000 Err Factor     0.554 Desired
    0.000 Sum          0.000 Transfer       0.000 Output
       0 Weights       0.000 Error          0.000 Current Error
 PE: 6
    1.000 Err Factor     0.514 Desired
    0.000 Sum          0.000 Transfer       0.000 Output
       0 Weights       0.000 Error          0.000 Current Error
 PE: 7
    1.000 Err Factor     0.467 Desired
    0.000 Sum          0.000 Transfer       0.000 Output
       0 Weights       0.000 Error          0.000 Current Error
 PE: 8
```

```
   1.000 Err Factor    0.416 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 9
   1.000 Err Factor    0.358 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 10
   1.000 Err Factor    0.295 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 11
   1.000 Err Factor    0.229 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 12
   1.000 Err Factor    0.164 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 13
   1.000 Err Factor    0.101 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 14
   1.000 Err Factor    0.044 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 15
   1.000 Err Factor   -0.006 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 16
   1.000 Err Factor   -0.045 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 17
   1.000 Err Factor   -0.070 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 18
   1.000 Err Factor   -0.085 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 19
   1.000 Err Factor   -0.091 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 20
   1.000 Err Factor   -0.096 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 21
   1.000 Err Factor   -0.108 Desired
```

```
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 22
   1.000 Err Factor  -0.124 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 23
   1.000 Err Factor  -0.145 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 24
   1.000 Err Factor  -0.168 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 25
   1.000 Err Factor  -0.193 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 26
   1.000 Err Factor  -0.218 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 27
   1.000 Err Factor  -0.245 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 28
   1.000 Err Factor  -0.268 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 29
   1.000 Err Factor  -0.287 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 30
   1.000 Err Factor  -0.304 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 31
   1.000 Err Factor  -0.319 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 32
   1.000 Err Factor  -0.331 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 33
   1.000 Err Factor  -0.341 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 34
   1.000 Err Factor  -0.351 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
```

```
    0 Weights      0.000 Error       0.000 Current Error
PE: 35
  1.000 Err Factor   -0.358 Desired
  0.000 Sum          0.000 Transfer     0.000 Output
    0 Weights      0.000 Error       0.000 Current Error
PE: 36
  1.000 Err Factor   -0.364 Desired
  0.000 Sum          0.000 Transfer     0.000 Output
    0 Weights      0.000 Error       0.000 Current Error
PE: 37
  1.000 Err Factor   -0.370 Desired
  0.000 Sum          0.000 Transfer     0.000 Output
    0 Weights      0.000 Error       0.000 Current Error
PE: 38
  1.000 Err Factor   -0.375 Desired
  0.000 Sum          0.000 Transfer     0.000 Output
    0 Weights      0.000 Error       0.000 Current Error
PE: 39
  1.000 Err Factor   -0.380 Desired
  0.000 Sum          0.000 Transfer     0.000 Output
    0 Weights      0.000 Error       0.000 Current Error
PE: 40
  1.000 Err Factor   -0.383 Desired
  0.000 Sum          0.000 Transfer     0.000 Output
    0 Weights      0.000 Error       0.000 Current Error
PE: 41
  1.000 Err Factor   -0.386 Desired
  0.000 Sum          0.000 Transfer     0.000 Output
    0 Weights      0.000 Error       0.000 Current Error
PE: 42
  1.000 Err Factor   -0.389 Desired
  0.000 Sum          0.000 Transfer     0.000 Output
    0 Weights      0.000 Error       0.000 Current Error
PE: 43
  1.000 Err Factor   -0.391 Desired
  0.000 Sum          0.000 Transfer     0.000 Output
    0 Weights      0.000 Error       0.000 Current Error
PE: 44
  1.000 Err Factor   -0.393 Desired
  0.000 Sum          0.000 Transfer     0.000 Output
    0 Weights      0.000 Error       0.000 Current Error
PE: 45
  1.000 Err Factor   -0.394 Desired
  0.000 Sum          0.000 Transfer     0.000 Output
    0 Weights      0.000 Error       0.000 Current Error
PE: 46
  1.000 Err Factor   -0.396 Desired
  0.000 Sum          0.000 Transfer     0.000 Output
    0 Weights      0.000 Error       0.000 Current Error
PE: 47
  1.000 Err Factor   -0.397 Desired
  0.000 Sum          0.000 Transfer     0.000 Output
    0 Weights      0.000 Error       0.000 Current Error
```

```
  PE: 48
     1.000 Err Factor    -0.399 Desired
     0.000 Sum           0.000 Transfer      0.000 Output
       0 Weights         0.000 Error         0.000 Current Error
  PE: 49
     1.000 Err Factor    -0.400 Desired
     0.000 Sum           0.000 Transfer      0.000 Output
       0 Weights         0.000 Error         0.000 Current Error
  PE: 50
     1.000 Err Factor    -0.401 Desired
     0.000 Sum           0.000 Transfer      0.000 Output
       0 Weights         0.000 Error         0.000 Current Error
  PE: 51
     1.000 Err Factor     0.731 Desired
     0.000 Sum           0.000 Transfer      0.000 Output
       0 Weights         0.000 Error         0.000 Current Error
Layer: Hidden1
     PEs: 18     Wgt Fields: 3              Sum: Sum
   Spacing: 5      F' offset: 0.00         Transfer: TanH
    Shape: Square                        Output: Direct
    Scale: 1.00    Low Limit: -9999.00    Error Func: standard
   Offset: 0.00    High Limit: 9999.00       Learn: Cum-Delta-Rule
  Init Low: -0.100  Init High: 0.100     L/R Schedule: hidden1
  Winner 1: None                         Winner 2: None
L/R Schedule: hidden1
  Recall Step          1       0       0       0       0
    Input Clamp      0.0000  0.0000  0.0000  0.0000  0.0000
    Firing Density 100.0000  0.0000  0.0000  0.0000  0.0000
    Temperature      0.0000  0.0000  0.0000  0.0000  0.0000
    Gain           1.0000  0.0000  0.0000  0.0000  0.0000
    Gain           1.0000  0.0000  0.0000  0.0000  0.0000
  Learn Step         10000   30000   70000  150000  310000
    Coefficient 1    0.3000  0.1500  0.0375  0.0023  0.0000
    Coefficient 2    0.4000  0.2000  0.0500  0.0031  0.0000
    Coefficient 3    0.1000  0.1000  0.1000  0.1000  0.1000
    Temperature      0.0000  0.0000  0.0000  0.0000  0.0000
  PE: 52
     1.000 Err Factor    0.000 Desired
     0.000 Sum           0.000 Transfer      0.000 Output
       51 Weights        0.000 Error         0.000 Current Error
  PE: 53
     1.000 Err Factor    0.000 Desired
     0.000 Sum           0.000 Transfer      0.000 Output
       51 Weights        0.000 Error         0.000 Current Error
  PE: 54
     1.000 Err Factor    0.000 Desired
     0.000 Sum           0.000 Transfer      0.000 Output
       51 Weights        0.000 Error         0.000 Current Error
  PE: 55
     1.000 Err Factor    0.000 Desired
     0.000 Sum           0.000 Transfer      0.000 Output
       51 Weights        0.000 Error         0.000 Current Error
  PE: 56
```

```
   1.000 Err Factor    0.000 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
     51 Weights        0.000 Error         0.000 Current Error
PE: 57
   1.000 Err Factor    0.000 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
     51 Weights        0.000 Error         0.000 Current Error
PE: 58
   1.000 Err Factor    0.000 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
     51 Weights        0.000 Error         0.000 Current Error
PE: 59
   1.000 Err Factor    0.000 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
     51 Weights        0.000 Error         0.000 Current Error
PE: 60
   1.000 Err Factor    0.000 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
     51 Weights        0.000 Error         0.000 Current Error
PE: 61
   1.000 Err Factor    0.000 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
     51 Weights        0.000 Error         0.000 Current Error
PE: 62
   1.000 Err Factor    0.000 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
     51 Weights        0.000 Error         0.000 Current Error
PE: 63
   1.000 Err Factor    0.000 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
     51 Weights        0.000 Error         0.000 Current Error
PE: 64
   1.000 Err Factor    0.000 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
     51 Weights        0.000 Error         0.000 Current Error
PE: 65
   1.000 Err Factor    0.000 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
     51 Weights        0.000 Error         0.000 Current Error
PE: 66
   1.000 Err Factor    0.000 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
     51 Weights        0.000 Error         0.000 Current Error
PE: 67
   1.000 Err Factor    0.000 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
     51 Weights        0.000 Error         0.000 Current Error
PE: 68
   1.000 Err Factor    0.000 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
     51 Weights        0.000 Error         0.000 Current Error
PE: 69
   1.000 Err Factor    0.000 Desired
```

```
    0.000 Sum          0.000 Transfer       0.000 Output
      51 Weights       0.000 Error          0.000 Current Error
Layer: Out
     PEs: 3     Wgt Fields: 3              Sum: Sum
  Spacing: 5     F' offset: 0.00          Transfer: TanH
    Shape: Square                        Output: Direct
    Scale: 1.00   Low Limit: -9999.00    Error Func: standard
   Offset: 0.00   High Limit: 9999.00        Learn: Cum-Delta-Rule
 Init Low: -0.100  Init High: 0.100     L/R Schedule: out
 Winner 1: None                         Winner 2: None
L/R Schedule: out
 Recall Step          1       0       0       0       0
  Input Clamp      0.0000  0.0000  0.0000  0.0000  0.0000
  Firing Density 100.0000  0.0000  0.0000  0.0000  0.0000
  Temperature      0.0000  0.0000  0.0000  0.0000  0.0000
  Gain           1.0000  0.0000  0.0000  0.0000  0.0000
  Gain           1.0000  0.0000  0.0000  0.0000  0.0000
 Learn Step         10000   30000   70000  150000  310000
  Coefficient 1    0.1500  0.0750  0.0188  0.0012  0.0000
  Coefficient 2    0.4000  0.2000  0.0500  0.0031  0.0000
  Coefficient 3    0.1000  0.1000  0.1000  0.1000  0.1000
  Temperature      0.0000  0.0000  0.0000  0.0000  0.0000
 PE: 70
   1.000 Err Factor   -0.050 Desired
   0.000 Sum          0.000 Transfer       0.000 Output
     19 Weights       0.000 Error          0.000 Current Error
 PE: 71
   1.000 Err Factor   -0.150 Desired
   0.000 Sum          0.000 Transfer       0.000 Output
     19 Weights       0.000 Error          0.000 Current Error
 PE: 72
   1.000 Err Factor    0.150 Desired
   0.000 Sum          0.000 Transfer       0.000 Output
     19 Weights       0.000 Error          0.000 Current Error
```

D.2. INITIAL NETWORK FOR CIRCUIT B

```
Title: Q=5.0/49in/18hid/2out
     Display Mode: Network                    Type: Hetero-Associative
    Display Style: default
  Control Strategy: backprop          L/R Schedule: backprop
        0 Learn          0 Recall          0 Layer
        1 Aux 1          0 Aux 2           0 Aux 3
L/R Schedule: backprop
  Recall Step          1       0       0       0       0
    Input Clamp      0.0000  0.0000  0.0000  0.0000  0.0000
    Firing Density 100.0000  0.0000  0.0000  0.0000  0.0000
    Temperature      0.0000  0.0000  0.0000  0.0000  0.0000
    Gain           1.0000  0.0000  0.0000  0.0000  0.0000
    Gain           1.0000  0.0000  0.0000  0.0000  0.0000
  Learn Step          5000       0       0       0       0
    Coefficient 1    0.9000  0.0000  0.0000  0.0000  0.0000
    Coefficient 2    0.6000  0.0000  0.0000  0.0000  0.0000
    Coefficient 3    0.0000  0.0000  0.0000  0.0000  0.0000
    Temperature      0.0000  0.0000  0.0000  0.0000  0.0000
IO Parameters
      Learn Data: File Rand. (eins6253) Binary Load
     Recall Data: File Seq. (eins6253)
     Result File: Desired Output, Output
   UserIO Program: userio
     I/P Ranges:   -1.0000,    1.0000
     O/P Ranges:   -0.6000,    0.6000
   I/P Start Col:        1          MinMax Table: eins6253
   O/P Start Col:       50          Number of Entries: 56
MinMax Table <eins6253>:
Col:      1       2       3       4       5       6
Min:  -0.0293  -0.0471  -0.0703  -0.1000  -0.1375  -0.1850
Max:    .115   .2008    .335   .5533   .9329   1.654
Col:      7       8       9      10      11      12
Min:  -0.2453  -0.3224  -0.4221  -0.5533  -0.7297  -1.1422
Max:   3.003    3.78   4.124   4.428   4.674   4.768
Col:     13      14      15      16      17      18
Min:  -1.7341  -2.4171  -3.2727  -4.2157  -4.5963  -3.9259
Max:   4.379   3.876   3.002   2.176   1.859   2.543
Col:     19      20      21      22      23      24
Min:  -2.9754  -2.2457  -1.7447  -1.3980  -1.1501  -0.9668
Max:   3.615   4.516   4.923   5.161   5.675   5.667
Col:     25      26      27      28      29      30
Min:  -0.8271  -0.7178  -0.6305  -0.5594  -0.5006  -0.4513
Max:   5.616   5.527   5.412   5.291   5.153   5.057
Col:     31      32      33      34      35      36
Min:  -0.4095  -0.3737  -0.3427  -0.3157  -0.2920  -0.2710
Max:   4.934   4.827   4.715   4.608   4.386   3.926
Col:     37      38      39      40      41      42
Min:  -0.2524  -0.2357  -0.2208  -0.2073  -0.1951  -0.1840
Max:   3.399    2.91   2.496   2.156   1.879   1.653
```

```
Col:      43       44       45       46       47       48
Min:  -0.1739   -0.1646   -0.1561   -0.1482   -0.1410   -0.1343
Max:   1.468     1.313     1.183     1.073     .9792     .8978
Col:      49       50       51       52       53       54
Min:  -0.1281   -0.6000   -0.6000   0.7000   0.7000    0.0000
Max:   .8271       .6        .6      1.3      1.3         0
Col:      55       56
Min:  0.0000   0.0000
Max:       0        0
Layer: 1
     PEs: 1     Wgt Fields: 2            Sum: Sum
  Spacing: 5     F' offset: 0.00       Transfer: Linear
    Shape: Square                     Output: Direct
    Scale: 1.00   Low Limit: -9999.00   Error Func: standard
   Offset: 0.00   High Limit: 9999.00      Learn: --None--
 Init Low: -0.100  Init High: 0.100    L/R Schedule: (Network)
 Winner 1: None                    Winner 2: None
 PE: Bias
    1.000 Err Factor     0.000 Desired
    0.000 Sum          1.000 Transfer       1.000 Output
       0 Weights       -7.508 Error         0.000 Current Error
Layer: In
     PEs: 49     Wgt Fields: 1            Sum: Sum
  Spacing: 5     F' offset: 0.00       Transfer: Linear
    Shape: Square                     Output: Direct
    Scale: 1.00   Low Limit: -9999.00   Error Func: standard
   Offset: 0.00   High Limit: 9999.00      Learn: --None--
 Init Low: -0.100  Init High: 0.100    L/R Schedule: (Network)
 Winner 1: None                    Winner 2: None
 PE: 2
    1.000 Err Factor     0.063 Desired
    0.000 Sum          0.000 Transfer       0.000 Output
       0 Weights       0.000 Error          0.000 Current Error
 PE: 3
    1.000 Err Factor     0.023 Desired
    0.000 Sum          0.000 Transfer       0.000 Output
       0 Weights       0.000 Error          0.000 Current Error
 PE: 4
    1.000 Err Factor    -0.032 Desired
    0.000 Sum          0.000 Transfer       0.000 Output
       0 Weights       0.000 Error          0.000 Current Error
 PE: 5
    1.000 Err Factor    -0.104 Desired
    0.000 Sum          0.000 Transfer       0.000 Output
       0 Weights       0.000 Error          0.000 Current Error
 PE: 6
    1.000 Err Factor    -0.200 Desired
    0.000 Sum          0.000 Transfer       0.000 Output
       0 Weights       0.000 Error          0.000 Current Error
 PE: 7
    1.000 Err Factor    -0.321 Desired
    0.000 Sum          0.000 Transfer       0.000 Output
       0 Weights       0.000 Error          0.000 Current Error
```

```
PE: 8
   1.000 Err Factor    -0.432 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 9
   1.000 Err Factor    -0.315 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 10
   1.000 Err Factor    -0.007 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 11
   1.000 Err Factor     0.516 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 12
   1.000 Err Factor     0.771 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 13
   1.000 Err Factor     0.071 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 14
   1.000 Err Factor    -0.379 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 15
   1.000 Err Factor    -0.606 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 16
   1.000 Err Factor    -0.720 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 17
   1.000 Err Factor    -0.793 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 18
   1.000 Err Factor    -0.839 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 19
   1.000 Err Factor    -0.870 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 20
   1.000 Err Factor    -0.894 Desired
   0.000 Sum           0.000 Transfer      0.000 Output
      0 Weights        0.000 Error         0.000 Current Error
PE: 21
```

```
   1.000 Err Factor    -0.913 Desired
   0.000 Sum            0.000 Transfer      0.000 Output
     0 Weights          0.000 Error         0.000 Current Error
PE: 22
   1.000 Err Factor    -0.924 Desired
   0.000 Sum            0.000 Transfer      0.000 Output
     0 Weights          0.000 Error         0.000 Current Error
PE: 23
   1.000 Err Factor    -0.933 Desired
   0.000 Sum            0.000 Transfer      0.000 Output
     0 Weights          0.000 Error         0.000 Current Error
PE: 24
   1.000 Err Factor    -0.943 Desired
   0.000 Sum            0.000 Transfer      0.000 Output
     0 Weights          0.000 Error         0.000 Current Error
PE: 25
   1.000 Err Factor    -0.948 Desired
   0.000 Sum            0.000 Transfer      0.000 Output
     0 Weights          0.000 Error         0.000 Current Error
PE: 26
   1.000 Err Factor    -0.952 Desired
   0.000 Sum            0.000 Transfer      0.000 Output
     0 Weights          0.000 Error         0.000 Current Error
PE: 27
   1.000 Err Factor    -0.955 Desired
   0.000 Sum            0.000 Transfer      0.000 Output
     0 Weights          0.000 Error         0.000 Current Error
PE: 28
   1.000 Err Factor    -0.958 Desired
   0.000 Sum            0.000 Transfer      0.000 Output
     0 Weights          0.000 Error         0.000 Current Error
PE: 29
   1.000 Err Factor    -0.960 Desired
   0.000 Sum            0.000 Transfer      0.000 Output
     0 Weights          0.000 Error         0.000 Current Error
PE: 30
   1.000 Err Factor    -0.962 Desired
   0.000 Sum            0.000 Transfer      0.000 Output
     0 Weights          0.000 Error         0.000 Current Error
PE: 31
   1.000 Err Factor    -0.965 Desired
   0.000 Sum            0.000 Transfer      0.000 Output
     0 Weights          0.000 Error         0.000 Current Error
PE: 32
   1.000 Err Factor    -0.966 Desired
   0.000 Sum            0.000 Transfer      0.000 Output
     0 Weights          0.000 Error         0.000 Current Error
PE: 33
   1.000 Err Factor    -0.968 Desired
   0.000 Sum            0.000 Transfer      0.000 Output
     0 Weights          0.000 Error         0.000 Current Error
PE: 34
   1.000 Err Factor    -0.969 Desired
```

```
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 35
   1.000 Err Factor   -0.970 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 36
   1.000 Err Factor   -0.971 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 37
   1.000 Err Factor   -0.970 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 38
   1.000 Err Factor   -0.967 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 39
   1.000 Err Factor   -0.964 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 40
   1.000 Err Factor   -0.961 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 41
   1.000 Err Factor   -0.957 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 42
   1.000 Err Factor   -0.954 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 43
   1.000 Err Factor   -0.951 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 44
   1.000 Err Factor   -0.947 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 45
   1.000 Err Factor   -0.945 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 46
   1.000 Err Factor   -0.942 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
     0 Weights      0.000 Error         0.000 Current Error
PE: 47
   1.000 Err Factor   -0.939 Desired
   0.000 Sum        0.000 Transfer      0.000 Output
```

```
      0 Weights      0.000 Error        0.000 Current Error
  PE: 48
     1.000 Err Factor   -0.936 Desired
     0.000 Sum          0.000 Transfer      0.000 Output
      0 Weights      0.000 Error        0.000 Current Error
  PE: 49
     1.000 Err Factor   -0.934 Desired
     0.000 Sum          0.000 Transfer      0.000 Output
      0 Weights      0.000 Error        0.000 Current Error
  PE: 50
     1.000 Err Factor   -0.932 Desired
     0.000 Sum          0.000 Transfer      0.000 Output
      0 Weights      0.000 Error        0.000 Current Error
Layer: Hidden1
     PEs: 18     Wgt Fields: 3              Sum: Sum
   Spacing: 5      F' offset: 0.00        Transfer: TanH
    Shape: Square                       Output: Direct
    Scale: 1.00    Low Limit: -9999.00    Error Func: standard
   Offset: 0.00    High Limit: 9999.00       Learn: Cum-Delta-Rule
  Init Low: -0.100  Init High: 0.100     L/R Schedule: hidden1
  Winner 1: None                        Winner 2: None
L/R Schedule: hidden1
  Recall Step          1       0       0       0       0
    Input Clamp      0.0000  0.0000  0.0000  0.0000  0.0000
    Firing Density 100.0000  0.0000  0.0000  0.0000  0.0000
    Temperature      0.0000  0.0000  0.0000  0.0000  0.0000
    Gain           1.0000  0.0000  0.0000  0.0000  0.0000
    Gain           1.0000  0.0000  0.0000  0.0000  0.0000
  Learn Step         20000   60000  140000  300000  620000
    Coefficient 1    0.3000  0.1500  0.0375  0.0023  0.0000
    Coefficient 2    0.4000  0.2000  0.0500  0.0031  0.0000
    Coefficient 3    0.1000  0.1000  0.1000  0.1000  0.1000
    Temperature      0.0000  0.0000  0.0000  0.0000  0.0000
  PE: 51
     1.000 Err Factor    0.000 Desired
     0.000 Sum          0.000 Transfer      0.000 Output
      50 Weights      0.000 Error        0.000 Current Error
  PE: 52
     1.000 Err Factor    0.000 Desired
     0.000 Sum          0.000 Transfer      0.000 Output
      50 Weights      0.000 Error        0.000 Current Error
  PE: 53
     1.000 Err Factor    0.000 Desired
     0.000 Sum          0.000 Transfer      0.000 Output
      50 Weights      0.000 Error        0.000 Current Error
  PE: 54
     1.000 Err Factor    0.000 Desired
     0.000 Sum          0.000 Transfer      0.000 Output
      50 Weights      0.000 Error        0.000 Current Error
  PE: 55
     1.000 Err Factor    0.000 Desired
     0.000 Sum          0.000 Transfer      0.000 Output
      50 Weights      0.000 Error        0.000 Current Error
```

PE: 56
1.000 Err Factor 0.000 Desired
0.000 Sum 0.000 Transfer 0.000 Output
50 Weights 0.000 Error 0.000 Current Error
PE: 57
1.000 Err Factor 0.000 Desired
0.000 Sum 0.000 Transfer 0.000 Output
50 Weights 0.000 Error 0.000 Current Error
PE: 58
1.000 Err Factor 0.000 Desired
0.000 Sum 0.000 Transfer 0.000 Output
50 Weights 0.000 Error 0.000 Current Error
PE: 59
1.000 Err Factor 0.000 Desired
0.000 Sum 0.000 Transfer 0.000 Output
50 Weights 0.000 Error 0.000 Current Error
PE: 60
1.000 Err Factor 0.000 Desired
0.000 Sum 0.000 Transfer 0.000 Output
50 Weights 0.000 Error 0.000 Current Error
PE: 61
1.000 Err Factor 0.000 Desired
0.000 Sum 0.000 Transfer 0.000 Output
50 Weights 0.000 Error 0.000 Current Error
PE: 62
1.000 Err Factor 0.000 Desired
0.000 Sum 0.000 Transfer 0.000 Output
50 Weights 0.000 Error 0.000 Current Error
PE: 63
1.000 Err Factor 0.000 Desired
0.000 Sum 0.000 Transfer 0.000 Output
50 Weights 0.000 Error 0.000 Current Error
PE: 64
1.000 Err Factor 0.000 Desired
0.000 Sum 0.000 Transfer 0.000 Output
50 Weights 0.000 Error 0.000 Current Error
PE: 65
1.000 Err Factor 0.000 Desired
0.000 Sum 0.000 Transfer 0.000 Output
50 Weights 0.000 Error 0.000 Current Error
PE: 66
1.000 Err Factor 0.000 Desired
0.000 Sum 0.000 Transfer 0.000 Output
50 Weights 0.000 Error 0.000 Current Error
PE: 67
1.000 Err Factor 0.000 Desired
0.000 Sum 0.000 Transfer 0.000 Output
50 Weights 0.000 Error 0.000 Current Error
PE: 68
1.000 Err Factor 0.000 Desired
0.000 Sum 0.000 Transfer 0.000 Output
50 Weights 0.000 Error 0.000 Current Error
Layer: Out

```
    PEs: 2      Wgt Fields: 3              Sum: Sum
 Spacing: 5     F' offset: 0.00         Transfer: TanH
  Shape: Square                        Output: Direct
  Scale: 1.00   Low Limit: -9999.00   Error Func: standard
 Offset: 0.00   High Limit: 9999.00       Learn: Cum-Delta-Rule
Init Low: -0.100  Init High: 0.100     L/R Schedule: out
Winner 1: None                        Winner 2: None
L/R Schedule: out
 Recall Step          1       0       0       0       0
  Input Clamp      0.0000  0.0000  0.0000  0.0000  0.0000
  Firing Density 100.0000  0.0000  0.0000  0.0000  0.0000
  Temperature      0.0000  0.0000  0.0000  0.0000  0.0000
  Gain           1.0000  0.0000  0.0000  0.0000  0.0000
  Gain           1.0000  0.0000  0.0000  0.0000  0.0000
 Learn Step         20000   60000  140000  300000  620000
  Coefficient 1    0.1500  0.0750  0.0188  0.0012  0.0000
  Coefficient 2    0.4000  0.2000  0.0500  0.0031  0.0000
  Coefficient 3    0.1000  0.1000  0.1000  0.1000  0.1000
  Temperature      0.0000  0.0000  0.0000  0.0000  0.0000
 PE: 69
   1.000 Err Factor    0.200 Desired
   0.000 Sum         0.000 Transfer      0.000 Output
     19 Weights      0.000 Error         0.000 Current Error
 PE: 70
   1.000 Err Factor    0.500 Desired
   0.000 Sum         0.000 Transfer      0.000 Output
     19 Weights      0.000 Error         0.000 Current Error
```